スケーラビリティ・ガイド

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

スケーラビリティ・ガイド
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

図一覧

テーブル一覧

1

InterSystems IRIS の拡張性の概要

今日のデータ・プラットフォームには、さまざまな作業負荷の処理が求められます。どのようなタイプの作業負荷が増加しても、データ・プラットフォームは、需要の増大に合わせて拡張可能であると同時に、企業が求めるパフォーマンス標準を維持し、ビジネスの中断を回避できなければなりません。

このドキュメントでは、InterSystems IRIS® データ・プラットフォームの拡張性について説明します。このドキュメントは、以下のような読者を対象としています。

- 企業内の固有のニーズに合わせて InterSystems IRIS の構成を計画および実装する業務に従事している。
- 企業における既存のニーズおよび今後のニーズに関連して、InterSystems IRIS を拡張するためのさまざまな機能を理解する必要がある。

ここでは、InterSystems IRIS データ・プラットフォームを拡張するための機能を検討し、エンタープライズ・データ・プラットフォームの拡張アプローチの一次評価に関するガイドラインを提供します。以下を含むそれぞれの機能については、この後の各トピックで詳しく説明します。

- 分散キャッシュによるユーザ数に応じた水平方向の拡張
- シャーディングによるデータ量に応じた水平方向の拡張

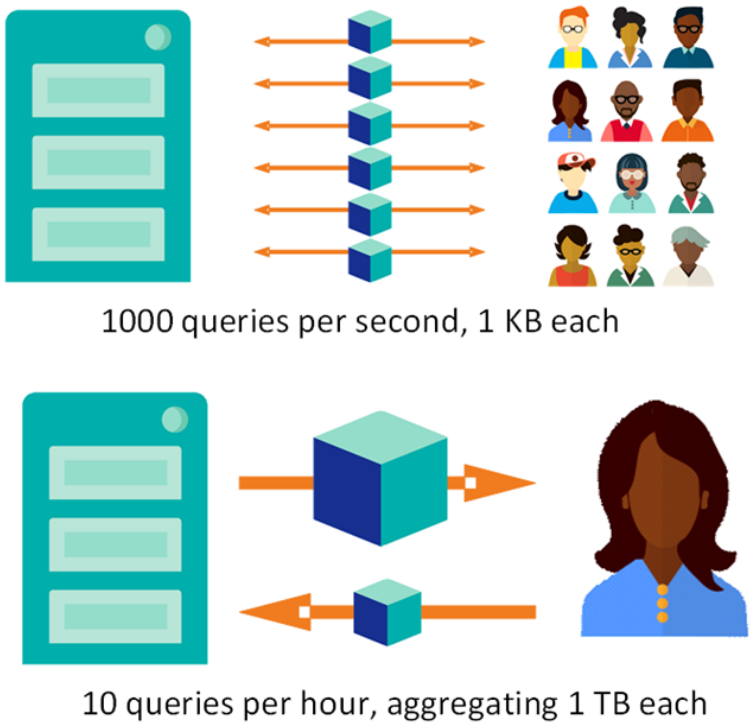
垂直方向の拡張の詳細は、"システム・リソースの計画と管理" および "InterSystems IRIS プラットフォームでの一般的なパフォーマンス強化" を参照してください。

1.1 拡張の問題

1,000 万人の患者を治療するのか、1 日数十億件の金融オーダーを処理するのか、星雲を追跡するのか、工場の 1,000 台のエンジンを監視するのかにかかわらず、何をするにしても、データ・プラットフォームは、現在の業務をサポートするだけでなく、需要の増大に合わせて拡張可能でなければなりません。ビジネス固有の作業負荷ごとに、それぞれが運用されるデータ・プラットフォームに対して異なる課題が突きつけられ、ビジネスが拡大するにつれて、その課題はさらに深刻になります。

例えば、以下の図に示す 2 つの状況を考えてみましょう。

図 1-1: 作業負荷の比較



どちらも高い作業負荷を示していますが、どちらの方が負荷が高いのか、それらの需要を満たすにはどのように拡張すればよいのかを判断するのは困難です。

データ・プラットフォームの作業負荷とそれらを調整するには何が必要であるかをより的確に理解するには、個々に調整できるコンポーネントへと作業負荷を分解します。これらの作業負荷を簡単に分解する 1 つの方法は、ユーザ数とデータ量というコンポーネントを分離することです。作業負荷には、最初の例のように、比較的小規模なデータベースを頻繁に操作する多数のユーザが含まれるものもあれば、2 つ目の例のように、1 人のユーザまたは 1 つのプロセスからの要求は少なくても、大規模なデータセットを対象とするものもあります。ユーザ数とデータ量を別個の課題と見なすことにより、さまざまな拡張オプションを評価できます(この分割は単純ですが、作業しやすく便利です。多数の小規模なデータ・ライターや少数の大規模なデータ・コンシューマを含むものなど、これが簡単に当てはまらない複雑な作業負荷の例も数多くあります)。

1.2 垂直方向の拡張

高い作業負荷を処理するための最も簡単な最初の方法は、スケールアップする、つまり、垂直方向の拡張性を利用することです。基本的には、作業負荷に対応できるように個々のサーバの性能を高めることを意味します。

図 1-2: 垂直方向の拡張

詳しく説明すると、垂直方向の拡張では、発生している作業負荷のボトルネックを軽減するハードウェア・コンポーネントを追加することにより、個々のサーバの処理能力を拡大する必要があります。例えば、現在のユーザ数やデータ量に必要な作業セットをキャッシュが処理できない場合は、マシンにメモリを追加できます。

一般に、垂直方向の拡張はよく理解されており、アーキテクチャも簡単です。十分なエンジニアリング・サポートがあれば、作業負荷の要件を満たすようにきめ細かく調整されたシステムを実現できます。ただし、以下のような制限があります。

- 100 を超える CPU コアとテラバイト単位のメモリを搭載した今日のサーバは非常に高性能ですが、その処理能力に関係なく、システムで受信接続について同時に作成および保持できるソケットの数は限られています。
- 高性能のハードウェアは高額であり、ソケットが不足したときには、システム全体をより大規模でより高価なものと交換するしか選択肢がない場合があります。
- 垂直方向の拡張を効果的に行うには、事前に慎重にサイジングする必要があります。これは、比較的静的なビジネスでは簡単な場合もありますが、急激に増加する作業負荷があるような動的な環境では、将来を予測するのが困難な場合があります。
- 垂直方向の拡張に弾性はありません。拡大した場合、作業負荷の変化により縮小が可能になっても、縮小することはできません。つまり、過剰な処理能力という代償を払うことになります。
- 垂直方向の拡張では、ハードウェアの処理能力の拡大に効果的かつ効率的に対応できる必要があるソフトウェアにストレスがかかります。例えば、アプリケーションで 32 のプロセスしか処理できない場合、128 コアに拡張してもほとんど役に立ちません。

InterSystems IRIS データ・プラットフォームの垂直方向の拡張の詳細は、"システム・リソースの計画と管理" および "InterSystems IRIS プラットフォームでの一般的なパフォーマンス強化" を参照してください。

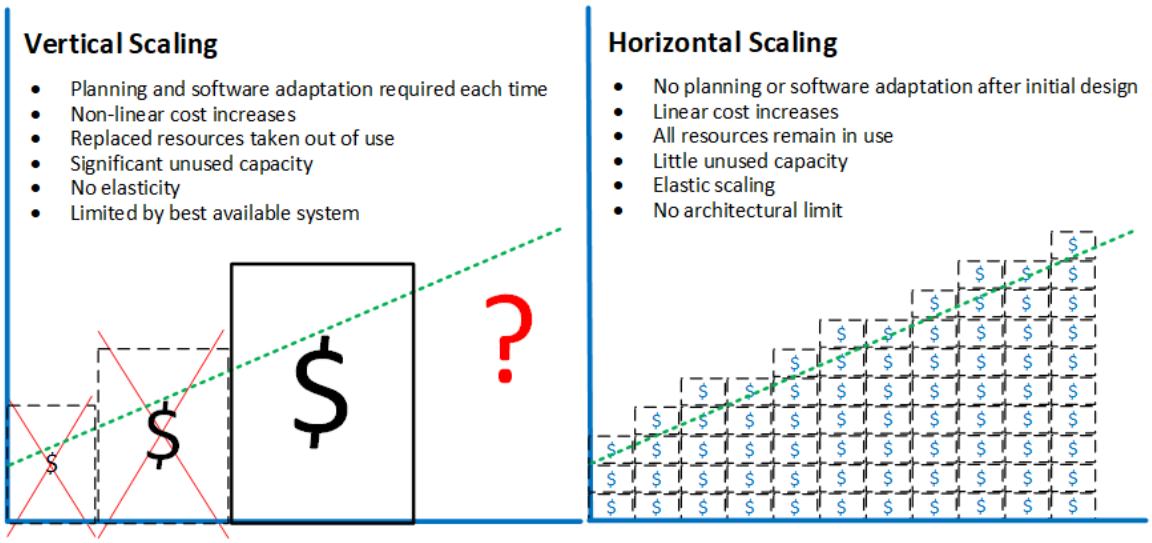
1.3 水平方向の拡張

回避できないハードウェア (または予算) 上限に達した場合など、垂直方向の拡張で完全なソリューションが実現されない場合、または垂直方向の拡張の代替手段として、複数の小型サーバをクラスタ化することでいくつかのデータ・プラットフォーム・テクノロジを水平方向に拡張することもできます。特定のコンポーネントを 1 台の高価なサーバに追加する代

わりに、この方法では、ボリュームの増大に伴って、より手ごろなサーバをクラスタに追加することで作業負荷をサポートできます。これは通常、1 台のサーバの作業負荷を小さく分割し、分割された個々の作業負荷を各クラスタ・ノードで処理できるようにすることを意味します。

水平方向の拡張は、多数の安価なコモディティ・システムから少数の高性能サーバまで、この範囲内のさまざまなハードウェアを使用して拡張できること、および垂直方向の拡張で必要な突然のデコミッションや置換ではなく、長期的にクラスタを拡張することで、拡張を徐々に行うことができることの両方の理由により、経済的に有利です。水平方向の拡張は、作業負荷の増加に伴って追加のノードをすばやくかつ簡単にプロビジョニングし、負荷が減少した場合はデコミッションできる、仮想インフラストラクチャやクラウド・インフラストラクチャにも非常に適しています。

図 1-3: 垂直方向の拡張での制限に対処する水平方向の拡張



その一方、水平クラスタでは、クラスタに含まれる複数のシステムに十分な帯域幅が提供されるように、ネットワーキング・コンポーネントに対してより大きな注意を払う必要があります。水平方向の拡張では、クラスタ全体での作業負荷の効率的な分散を完全にサポートするために、InterSystems IRIS のようなはるかに高度なソフトウェアも必要です。InterSystems IRIS は、ユーザ数の増加とデータ量の増加の両方に対応する拡張性を提供することにより、これを実現します。

1.3.1 ユーザ数に応じた水平方向の拡張

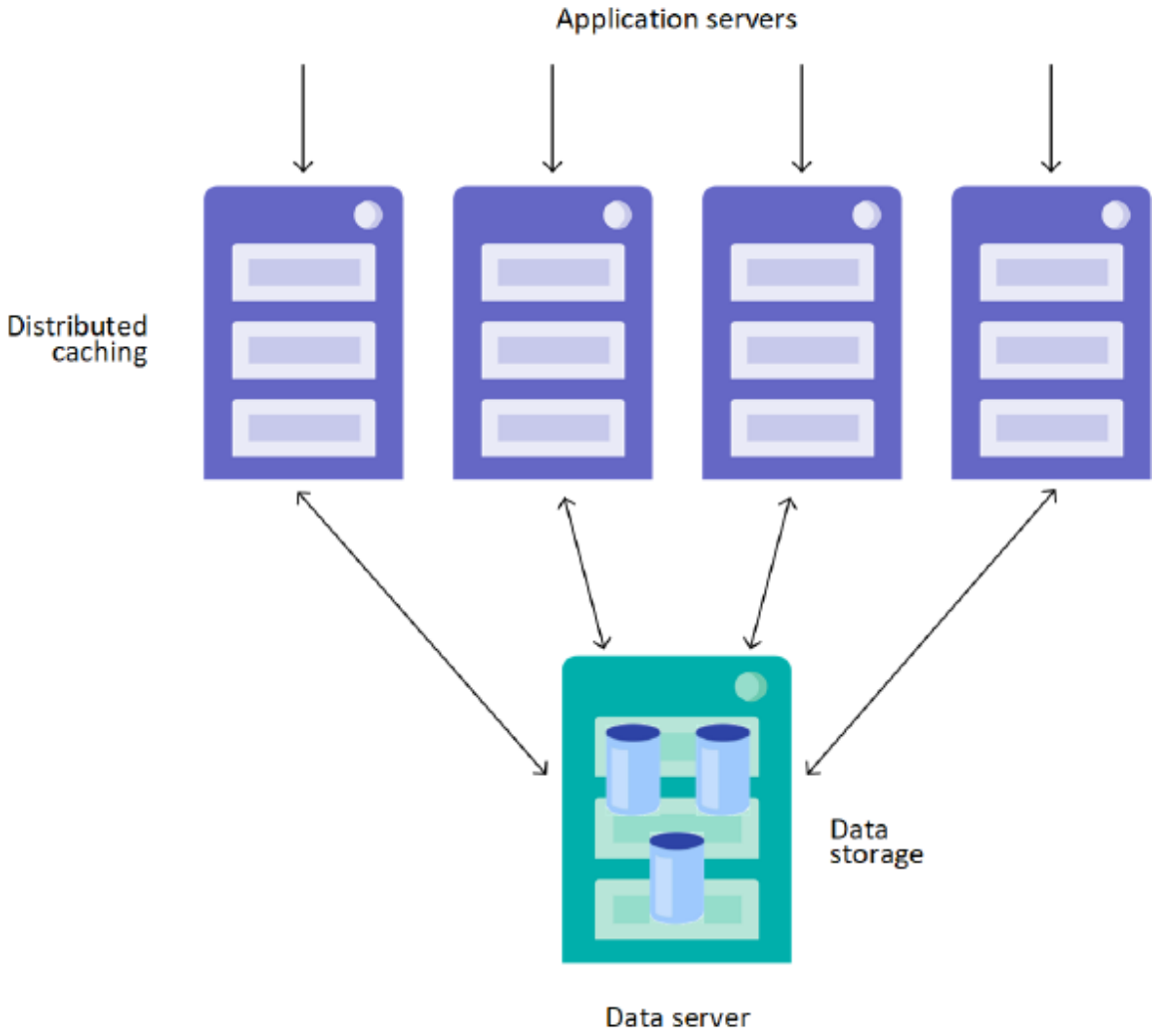
ユーザ数が増加し、許容できるコストの 1 つのシステムでは対処できなくなった場合、水平方向に拡張するにはどうすればよいのでしょうか。簡単に言えば、個々のユーザを、それぞれの要求を処理する異なるクラスタ・ノードに接続して、ユーザ作業負荷を分配します。

図 1-4: ユーザ作業負荷の分配

これを行うには、ロード・バランサを使用してラウンドロビン方式でユーザを分散します。ただし、同様の要求を持つユーザ（複数のアプリケーションが使用されている場合に特定のアプリケーションを使用するユーザなど）を同じノード上にグループ化すると、分散キャッシュによってユーザが相互のキャッシュを利用できるため、より効果的です。

InterSystems IRIS は、データ・サーバの前にあるアプリケーション・サーバの層全体にわたってユーザを分配するエンタープライズ・キャッシュ・プロトコル（ECP）によりサポートされているアーキテクチャ・ソリューションである分散キャッシュを使用して、効果的にこれを行うことができます。各アプリケーション・サーバは、独自のキャッシュを使用してユーザのクエリおよびトランザクションを処理しますが、データはすべてデータ・サーバに格納され、アプリケーション・サーバのキャッシュは自動的に同期された状態に維持されます。各アプリケーション・サーバが独自のキャッシュ内にそれぞれ独自の作業セットを保持するため、より多くのサーバを追加すると、より多くのユーザを処理できます。

図 1-5: InterSystems IRIS 分散キャッシュ・クラスタ



分散キャッシュは、ユーザおよびアプリケーション・コードに対して完全に透過的です。

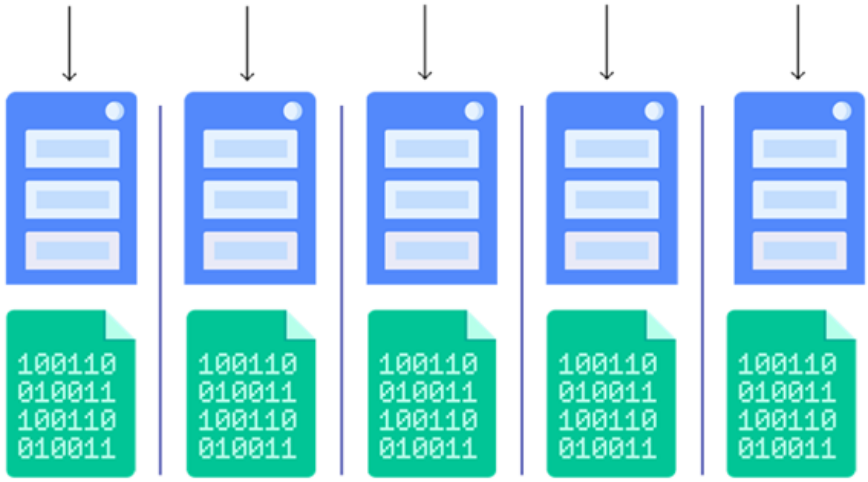
ユーザ数に応じた InterSystems IRIS データ・プラットフォームの水平方向の拡張の詳細は、"分散キャッシュによるユーザ数に応じた水平方向の拡張" を参照してください。

1.3.2 データ量に応じた水平方向の拡張

今日の企業のニーズを満たすために必要なデータ量は、非常に大きい場合があります。さらに重要なことに、クエリが繰り返し実行されると、作業セットが大きくなり、サーバのキャッシュに収まらなくなることがあります。これは、作業セットの一部しかキャッシュに保持できず、ディスクの読み取りがはるかに頻繁になり、クエリのパフォーマンスに重大な影響が生じることを意味します。

ユーザ数の場合と同様に、作業負荷を複数のサーバ間で分配することにより、データ量に応じて水平方向に拡張できます。そのためには、データを分割します。

図 1-6: データ作業負荷の分配



InterSystems IRIS は、シャーディング機能を使用してこれを行います。InterSystems IRIS シャード・クラスタは、データの格納および対応するキャッシュを複数のサーバ間で分割して、クエリおよびデータ取り込みに応じた水平方向の拡張を実現しながら、きわめて効率的にリソースを使用することにより、インフラストラクチャの価値を最大化します。

基本的なシャード・クラスタでは、シャード・テーブルが、シャードと呼ばれるほぼ均等な行セットに水平方向に分割され、複数のデータ・ノード間で分散されます。例えば、1 億の行があるテーブルが 4 台のデータ・ノード間で分割された場合、約 2,500 万の行を含むシャードが各サーバに格納されます。シャード化されていないテーブルは、全体が最初に構成されたデータ・ノードに配置されます。

シャード・テーブルに対するクエリは、複数のシャードローカル・クエリに分解され、データ・ノード上で並列で実行されます。その後、結果が結合され、ユーザに返されます。さらに、この分散データ・レイアウトはデータの並列ロードに利用したり、サードパーティ・フレームワークと共に利用したりすることもできます。

並列処理に加えて、シャーディングでは、キャッシュが分割されるため、クエリのパフォーマンスが向上します。各データ・ノードは、サーバに格納されているデータに対するシャードローカル・クエリに独自のキャッシュを使用するため、シャード・データを処理するクラスタのキャッシュは、すべてのデータ・ノードのキャッシュの合計とほぼ同じ大きさになります。データ・ノードの追加は、より多くのデータを処理するために専用キャッシュを追加することを意味します。

アプリケーション・サーバ・アーキテクチャと同様に、シャーディングは、ユーザおよびアプリケーションに対して完全に透過的です。

シャーディングには、使用可能なソリューションの幅が大幅に広がる、以下のような追加オプションがあります。

- ミラーリング

  InterSystems IRIS のミラーリングを使用して、データ・ノードの高可用性を実現できます。

- 計算ノード

  常に大量のデータが取り込まれるような状況でも、遅延が小さいことが求められる高度な使用事例では、計算ノードを追加して、クエリを処理するための透過的なキャッシュ層を提供できます。計算ノードはクエリの実行のみをサポートし、割り当てられているデータ・ノードのシャード・データ（および、必要に応じてシャード化されていないデータ）をキャッシュします。クラスタに計算ノードが含まれている場合、計算ノードでは読み取り専用クエリが自動的に実行され、データ・ノードではすべての書き込み操作（挿入、更新、削除、および DDL 操作）が実行されます。このクエリの作業負荷とデータ取り込みの分離により、それら両方のパフォーマンスが向上し、1 つのデータ・ノードに複数の計算ノードを割り当てることで、クエリ・スループットとクラスタのパフォーマンスを向上させることができます。

データ量に応じた InterSystems IRIS データ・プラットフォームの水平方向の拡張の詳細は、"シャーディングによるデータ量に応じた水平方向の拡張" を参照してください。

1.3.3 水平方向拡張構成の自動導入

InterSystems IRIS データ・プラットフォームではクラスタの自動導入方法がいくつか提供されており、複雑な水平クラスタ構成の場合は特に、導入プロセスを大幅に簡素化することが可能です。クラスタの自動導入の詳細は、“シャーディングによるデータ量に応じた水平方向の拡張” の章の ″クラスタの自動導入方法″ を参照してください。

1.4 拡張ソリューションに関する作業負荷の評価

このドキュメントの後続の章では、InterSystems IRIS を拡張するための個々の機能について詳しく説明します。データ・プラットフォームの拡張プロセスを開始する前に、各章を参照してください。ただし、以下のテーブルでは、この章の概要をまとめ、現在の状況に最も有効であると思われる拡張アプローチに関する一般的なガイドラインを提供します。

テーブル 1-1: InterSystems IRIS の拡張の長所と短所

拡張アプローチ	長所	短所
垂直方向	アーキテクチャのシンプルさ 作業負荷に合わせてきめ細かく調整されたハードウェア	非線形的な価格/パフォーマンス比率 永続ハードウェアの制限 慎重な初期サイジングが必要 単方向の拡張のみ
水平方向	より非線形的な価格/パフォーマンス比率 仮想およびクラウドベースのコモディティ・システムを利用可能 弾力性に優れた拡張	ネットワーキングを重視

テーブル 1-2: InterSystems IRIS の垂直方向の拡張ソリューション

状況	考えられるソリューション
多数のユーザによる大量のクエリ：処理能力の不足、クエリの量に対して不十分なスループット。	CPU コアを追加します。 クエリの並列実行を活用して、大規模なデータセットにわたるクエリに多数のコアを利用します。
大量のデータ：メモリ不足、作業セットに対して不十分なデータベース・キャッシュ。	メモリを追加し、キャッシュ・サイズを大きくして、より大きなメモリを利用します。 クエリの並列実行を活用して、多数のコアを利用します。
他の不十分な処理能力：ネットワーク帯域幅など、他の領域におけるボトルネック。	ボトルネックの原因となっている可能性がある他のリソースを増やします。

テーブル 1-3: InterSystems IRIS の水平方向の拡張ソリューション

状況	考えられるソリューション
多数のユーザによる大量のクエリ：多数のユーザからの頻繁なクエリ。	アプリケーション・サーバ構成（分散キャッシュ）を導入します。
大量のデータ：以下のいくつかの組み合わせ ・ 大量または高頻度（あるいはその両方）のデータ取り込み ・ 大規模なデータ・セット ・ 大量のデータ処理を含む複雑なクエリ（"シャーディングの効果の評価" を参照）	シャード・クラスタ（分割されたデータおよび分割されたキャッシュ）を導入し、可能であれば、データ取り込みからクエリを分離し、クエリ・スループットを向上させる計算ノードを追加します（"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" を参照）。

1.5 InterSystems IRIS プラットフォームでの一般的なパフォーマンス強化

以下の情報は、InterSystems IRIS の導入のパフォーマンスを向上させる際に役立ちます。

- 同時マルチスレッディング

  ほとんどの状況では、物理サーバ内、または仮想環境内のハイパーバイザ層でパフォーマンスを向上させるには、Intel のハイパースレッディングや AMD の同時マルチスレッディング（SMT）の使用をお勧めします。仮想環境において、ハイパースレッディングや SMT の無効化が必要な場合もありますが、これらは指定されたアプリケーションに固有の例外的なケースです。

  IBM AIX® の場合、IBM Power プロセッサは 1 コアあたり 2、4、8 スレッドという複数の SMT レベルを提供します。最新の IBM Power9 および Power10 プロセッサでは、SMT-8 が InterSystems IRIS で最もよく使用されているレベルです。ただし、以前の世代の Power7 や Power8 プロセッサでは特に、指定のアプリケーションには SMT-2 や SMT-4 の方が適している場合もあります。特定の導入で理想的な SMT レベルを判断するには、アプリケーションのベンチマークがお勧めです。

- セマフォ割り当て

  既定では、InterSystems IRIS は、セットあたりのセマフォの数を最大にすることで、最小数のセマフォ・セットを割り当てます（"インターシステムズ製品のセマフォ" を参照）。ただし、non-uniform memory access（NUMA）アーキテクチャの Linux システムでのパフォーマンスに対しては、これは理想的ではないという情報もあります。

  これに対処するため、構成パラメータ・ファイル（CPF）の semsperset パラメータを使用して、セットあたりのセマフォをより小さい数に指定できます。既定では、semsperset は 0（既定の動作を指定）に設定されます。最も効果的な設定を決定するには経験が必要です。Linux/NUMA システムで InterSystems IRIS を導入した場合、初期値である 250 を試すことをお勧めします。

2

分散キャッシュによるユーザ数に応じた水平方向の拡張

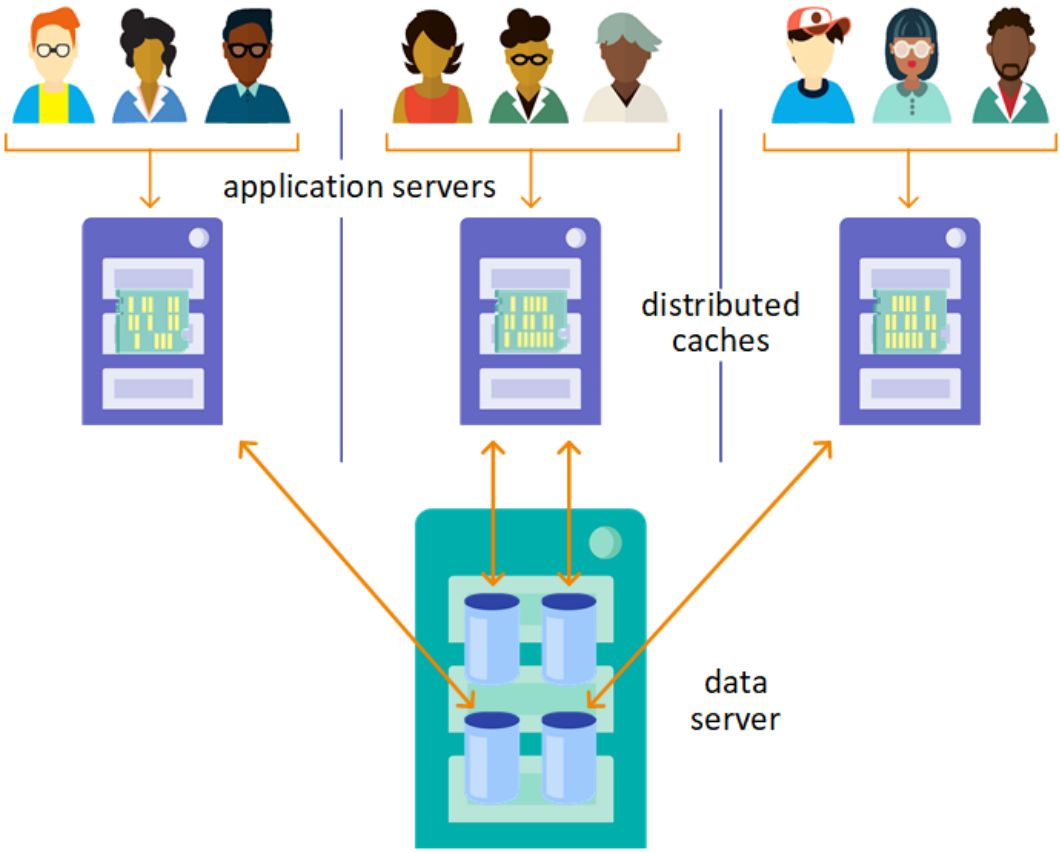
InterSystems IRIS データ・プラットフォームを拡張して作業負荷の要件を満たすために、垂直方向の拡張だけでは不十分な場合は、アーキテクチャが簡単で、アプリケーションに透過的であり、低コストの水平方向の拡張アプローチである、分散キャッシュを検討してください。

2.1 分散キャッシュの概要

InterSystems IRIS 分散キャッシュ・アーキテクチャでは、データ・サーバの前にあるアプリケーション・サーバの層全体にわたってアプリケーション・ロジックとキャッシュの両方を分散することでユーザ数に応じて水平方向に拡張し、その層全体にわたるユーザの分配を実現しています。各アプリケーション・サーバはユーザの要求を処理し、自動的にデータ・サーバと同期された状態に維持される独自のデータベース・キャッシュを保持します。一方、データ・サーバはすべてのデータの格納と管理を処理します。アプリケーション・サーバとデータ・サーバ間の中断された接続は、停止の長さに応じて自動的に回復またはリセットされます。

分散キャッシュを使用すると、各アプリケーション・サーバは、データの作業セットをそれぞれ独自に保持できます。これにより、単一のサーバ上に作業セット全体を格納するために、費用のかかる大量のメモリを確保する必要がなくなり、安価なアプリケーション・サーバを追加して、より多くのユーザを処理することができます。分散キャッシュは、アプリケーションが、使用可能な CPU 容量によって制限される場合にも役立ちます。この場合も、単一のサーバ用に高価なプロセッサを購入するのではなく、コモディティ・アプリケーション・サーバを追加することで容量が増えます。

図 2-1: 分散キャッシュ・クラスタ



このアーキテクチャは、InterSystems IRIS データ・プラットフォームのコア・コンポーネントである Enterprise Cache Protocol (ECP) をアプリケーション・サーバとデータ・サーバの間の通信に使用することで実現します。

分散キャッシュ・アーキテクチャおよびアプリケーション・サーバ層は、ユーザおよびアプリケーション・コードに対して完全に透過的です。アプリケーション・サーバを追加することにより、データを提供する既存のスタンドアロン InterSystems IRIS インスタンスを、クラスタのデータ・サーバへと簡単に変換できます。

以下のセクションでは、分散キャッシュについて詳しく説明します。

- 分散キャッシュ・アーキテクチャ
- ECP の機能
- ECP リカバリ
- 分散キャッシュと高可用性

2.1.1 分散キャッシュ・アーキテクチャ

分散キャッシュ・アーキテクチャをよりよく理解するために、InterSystems IRIS によってデータがどのように格納され、アクセスされるのかについて、以下の情報を確認してください。

- InterSystems IRIS は、データベースと呼ばれるローカル・オペレーティング・システムのファイルにデータを格納します。InterSystems IRIS インスタンスは、(通常) 複数のデータベースを持っています。
- InterSystems IRIS アプリケーションは、1 つ以上のデータベースに格納されたデータの論理ビューを提供する、ネームスペースを使用してデータにアクセスします。InterSystems IRIS インスタンスは、(通常) 複数のネームスペースを持っています。

- それぞれの InterSystems IRIS インスタンスは、データベース・キャッシュを保持します。データベース・キャッシュとは、ローカルの共有メモリ・バッファで、データベースから取得されるデータをキャッシュするために使用します。これにより、同じクエリの繰り返しインスタンスは、ストレージではなくメモリから結果を取得できるので、パフォーマンス上のメリットが非常に大きくなります。

分散キャッシュ・クラスタのアーキテクチャは、概念的には単純であり、これらの要素を以下の方法で使用します。

- InterSystems IRIS インスタンスをアプリケーション・サーバにするには、別のインスタンスをリモート・サーバとして追加した後、そのデータベースのいずれかまたはすべてをリモート・データベースとして追加します。これにより、2 つ目のインスタンスが 1 つ目のインスタンスのデータ・サーバになります。
- アプリケーション・サーバ上のローカル・ネームスペースを、データ・サーバ上のリモート・データベースにマッピングする方法は、ローカル・データベースにマッピングする場合と同じです。ローカル・データベースとリモート・データベースの違いは、アプリケーション・サーバ上のネームスペースに対してクエリを実行するアプリケーションにとって完全に透過的であることです。
- アプリケーション・サーバは、ローカル・データベースのみを使用する場合と同じ方法で、独自のデータベース・キャッシュを維持します。ECP は、複数の InterSystems IRIS インスタンス間でデータ、ロック、実行可能コードを効率的に共有し、アプリケーション・サーバのキャッシュをデータ・サーバと同期させます。

実際には、複数のアプリケーション・サーバの分散キャッシュ・クラスタとデータ・サーバは、次のように動作します。

- データ・サーバは、データの格納、更新、および提供を引き続き行います。また、データ・サーバは、ユーザが古いデータを受け取ったり、保持したりすることがないようにアプリケーション・サーバのキャッシュを同期して整合性を維持し、クラスタ全体のロックを管理します。
- データに対する各クエリはさまざまなアプリケーション・サーバのいずれかのネームスペースで行われ、各アプリケーション・サーバはそれぞれ独自のデータベース・キャッシュを使用して、受け取った結果をキャッシュします。その結果、キャッシュされたデータのセット全体がこれらの個々のキャッシュ間で分散されます。複数のデータ・サーバがある場合、アプリケーション・サーバは、要求されたデータを格納しているデータ・サーバに自動的に接続します。また、各アプリケーション・サーバはそのデータ・サーバ接続を監視し、接続が中断した場合は、接続を回復しようとします。
- ロード・バランサによってユーザの要求をラウンドロビン方式でアプリケーション・サーバ間で分散することもできますが、最も効果的なアプローチでは、同様の要求を持つユーザを同じアプリケーション・サーバに送ることで、キャッシュ効率を高め、分散キャッシュを最大限に活用します。例えば、医療アプリケーションでは、あるクエリ・セットを実行する医療ユーザを 1 つのアプリケーション・サーバにグループ化し、異なるセットを実行するフロントデスク・スタッフを別のサーバにグループ化できます。クラスタで複数のアプリケーションを処理する場合は、各アプリケーションのユーザを別個のアプリケーション・サーバに送ることができます。以下の各図は、単一 InterSystems IRIS インスタンスと、ユーザ接続がこのように分散されるクラスタを比較したものです。(負荷分散のユーザ要求は、状況によっては有害な影響を与える場合があります。詳細は、"負荷分散のユーザ要求の影響の評価" を参照してください。)
- クラスタの再構成や動作の変更を別個に行うことなく、クラスタ内のアプリケーション・サーバの数を増減できるため、ユーザ数の増加に伴って、簡単に拡張することが可能です。

図 2-2: 単一 InterSystems IRIS インスタンス上のローカル・ネームスペースにマッピングされたローカル・データベース

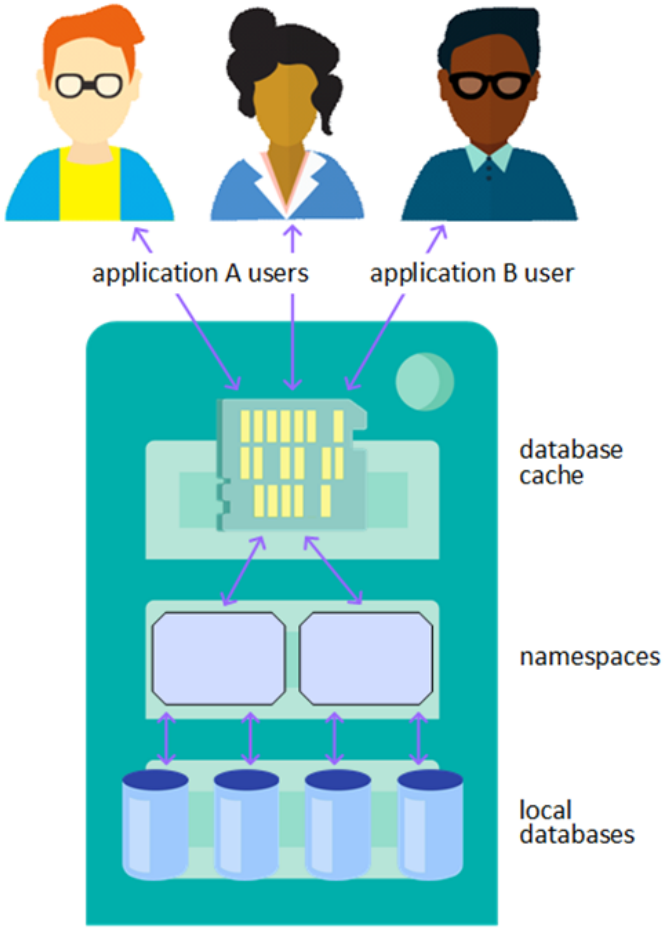
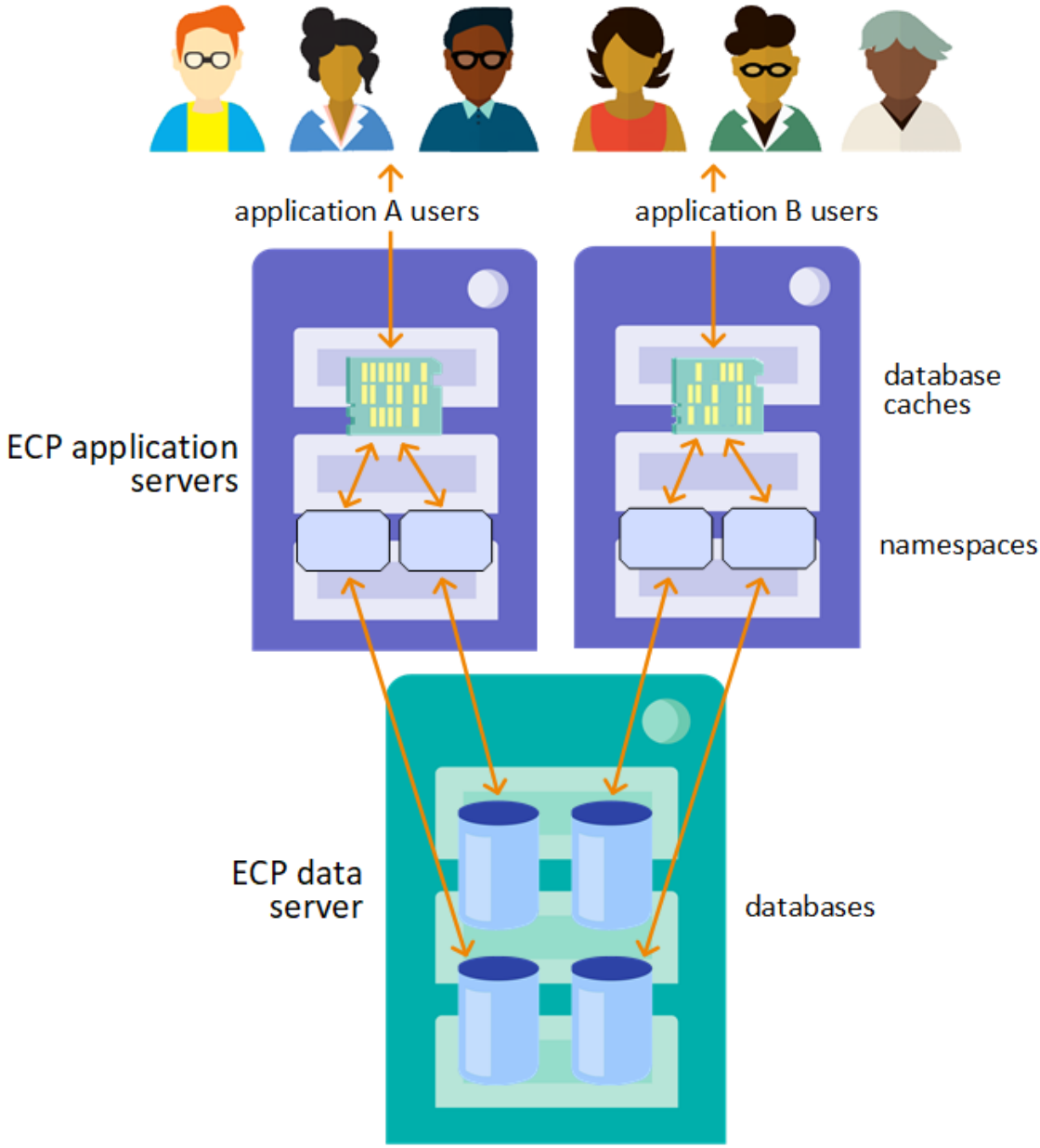


図 2-3: 分散キャッシュ・クラスタ内のアプリケーション・サーバ上のネームスペースにマッピングされたデータ・サーバ上のリモート・データベース



分散キャッシュ・クラスタのデータ・サーバには、以下の役割があります。

- ローカル・データベースにデータを格納します。
- アプリケーション・サーバが古いデータを参照しないように、アプリケーション・サーバのデータベース・キャッシュをデータベースと同期させます。
- ネットワーク経由のロックの分散を管理します。
- すべてのアプリケーション・サーバ接続の状態を監視し、一定時間接続が中断された場合には対応策を実行します("ECP リカバリ" を参照)。

分散キャッシュ・クラスタの各アプリケーション・サーバには、以下の役割があります。

- アプリケーションが、データ・サーバに格納されたデータを要求するたびにそのデータ・サーバへの接続を確立します。

- ネットワークから取得したデータをキャッシュ内で維持します。
- データ・サーバへのすべての接続の状態を監視し、一定時間接続が中断された場合には対応策を実行します ("ECP リカバリ" を参照)。

注釈　分散キャッシュ・クラスタには、複数のデータ・サーバを含めることができます (ただし、これが行われるのはまれです)。InterSystems IRIS インスタンスは、データ・サーバとアプリケーション・サーバの両方の機能を同時に果たすことができますが、アプリケーション・サーバとして受信するデータのデータ・サーバとしては機能しません。

2.1.2 ECP の機能

ECP は、次の機能を提供することにより、分散キャッシュ・アーキテクチャをサポートしています。

- 自動、フェイルセーフ処理。ECP は、一度構成されると、アプリケーション・サーバとデータ・サーバ間の接続を自動的に確立、維持し、アプリケーション・サーバとデータ・サーバ・インスタンス間の接続が (計画的または計画外に) 切断された場合、回復しようとします ("ECP リカバリ" を参照)。また ECP では、データ・サーバのフェイルオーバーが発生しても、その前後で実行中のアプリケーションの状態を維持することもできます ("分散キャッシュと高可用性" を参照)。

  これらの機能により、アプリケーションでデータを引き続き利用できるだけでなく、分散キャッシュ・クラスタの管理が容易になります。例えば、アプリケーション・サーバのインスタンスで操作を実行することなく、一時的にデータ・サーバをオフラインにしたり、計画的なメンテナンスの一部としてフェイルオーバーさせたりすることが可能です。
- 混在するネットワーク : 分散キャッシュ・クラスタ内の InterSystems IRIS システムは、さまざまなハードウェアおよびオペレーティング・システム・プラットフォームで実行できます。データ形式の変換が必要な場合、ECP が自動的にそれを管理します。
- TCP/IP ベースの堅牢な Transport 層 : ECP は構成と維持が容易な、標準的な TCP/IP プロトコルをデータ転送に使用します。
- ネットワーク帯域幅の効率的な使用 : ECP は、高性能ネットワーキング・インフラストラクチャをフルに活用します。

2.1.3 ECP リカバリ

ECP は、アプリケーション・サーバとデータ・サーバの間の接続の中断から自動的に回復するように設計されています。このような中断が発生した場合、ECP はリカバリ・プロトコルを実行します。リカバリ・プロトコルは、障害の性質と構成されたタイムアウト間隔によって異なります。その結果は、接続が回復されるか、何も発生しなかったかのようにアプリケーションのプロセスを続行できるか、またはリセットされ、トランザクションを強制的にロールバックし、アプリケーションのプロセスが再構築されます。

ECP 接続の詳細は、"分散アプリケーションの監視" を参照してください。ECP リカバリの詳細は、"ECP リカバリ・プロトコル" および "ECP リカバリ・プロセス、保証、および制限" を参照してください。

2.1.4 分散キャッシュと高可用性

ECP リカバリによって、データ・サーバへの中断されたアプリケーション・サーバ接続が処理されている間、分散キャッシュ・クラスタ内のアプリケーション・サーバは、データ・サーバのフェイルオーバーが発生しても、その前後で実行中のアプリケーションの状態が維持されるように設計されています。アプリケーション・アクティビティとフェイルオーバー・メカニズムの特性に応じて、フェイルオーバーが完了するまで一時停止する場合がありますが、その後ワークフローを中断することなく動作を続行できます。

スタンドアロンの InterSystems IRIS インスタンスと同じように、高可用性を実現するためにデータ・サーバをミラーリングし、フェイルオーバーが発生した場合に接続を自動的にバックアップにリダイレクトするようにアプリケーション・サーバを設定することもできます。(アプリケーション・サーバはデータを格納しないため、ミラーリングする必要はなく、ミラーリング

することはありません)。分散キャッシュ・クラスタでのミラーリングの使用方法の詳細は、"ミラーへの ECP 接続の構成"を参照してください。

"高可用性を実現するためのフェイルオーバー方法" で詳しく説明されているその他のフェイルオーバー方法は、分散キャッシュ・クラスタでも使用できます。データ・サーバに使用するフェイルオーバーの方法に関係なく、アプリケーション・サーバは、フェイルオーバーに続いて再接続してその状態を回復し、障害発生前の場所からアプリケーションのプロセスを続行できます。

2.2 分散キャッシュ・クラスタの導入

InterSystems IRIS 分散キャッシュ・クラスタは、1 つ以上のアプリケーション・サーバ・インスタンスにデータを提供するデータ・サーバ・インスタンスで構成され、これらのアプリケーション・サーバ・インスタンスから、アプリケーションにデータが提供されます。InterSystems IRIS データ・プラットフォームでは、分散キャッシュ・クラスタの自動導入方法がいくつか提供されています。このセクションで紹介している手順は、管理ポータルを使用してクラスタを手動で導入するためのものです。導入後のクラスタのセキュリティについては、"分散キャッシュ・クラスタのセキュリティ" を参照してください。

重要　一般に、分散キャッシュ・クラスタでの InterSystems IRIS インスタンスは、どのアプリケーション・サーバもデータ・サーバより後のバージョンでない限り、さまざまなバージョンがあり得ます。バージョンの互換性に関する重要な要件および制限事項は、"インターシステムズのサポート対象プラットフォーム" の "ECP の相互運用性" を参照してください。

データ・サーバとアプリケーション・サーバのホストは異なるオペレーティング・システムまたはエンディアン (またはその両方) とすることができますが、分散キャッシュ・クラスタ内の InterSystems IRIS インスタンスはすべて同じロケールを使用する必要があります。ロケールの構成の詳細は、"管理ポータルの NLS 設定ページの使用法" を参照してください。

注釈　複数のアプリケーション・サーバにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明は、"負荷分散、フェイルオーバー、ミラー構成" を参照してください。

メモリ管理および拡張、CPU サイジングおよび拡張、その他の考慮事項を含む、パフォーマンス計画の重要な説明は、"システム・リソースの計画と管理" を参照してください。

最も一般的な分散キャッシュ・クラスタ構成では、ホストごとに 1 つの InterSystems IRIS インスタンスと、インスタンスごとに 1 つのクラスタ・ロールがあります (つまり、データ・サーバまたはアプリケーション・サーバのいずれか)。次のセクションで説明する自動導入方法の 1 つを使用する場合、この構成が唯一のオプションです。管理ポータルの使用について記載されている手順も、この構成を前提としています。

HealthShare Health Connect では分散キャッシュはサポートされません。

2.2.1 クラスタの自動導入方法

このセクションで概要を説明する手動手順のほかにも、InterSystems IRIS データ・プラットフォームでは、導入後に完全に機能する分散キャッシュ・クラスタの自動導入方法が 2 つ提供されています。

2.2.1.1 InterSystems Kubernetes Operator (IKO) を使用した分散キャッシュ・クラスタの導入

Kubernetes は、コンテナ化されたワークロードとサービスの導入、拡張、および管理を自動化するためのオープンソースのオーケストレーション・エンジンです。導入するコンテナ化されたサービスと、そのサービスを管理するポリシーを定義すると、Kubernetes は、必要なリソースを可能な限り最も効率的な方法で透過的に提供します。また、導入が指定値から外れた場合は導入を修復またはリストアするほか、拡張を自動またはオンデマンドで行います。InterSystems Kubernetes Operator (IKO) は、IrisCluster カスタム・リソースで Kubernetes API を拡張します。このリソースは、InterSys-

tems IRIS のシャード・クラスタ、分散キャッシュ・クラスタ、またはスタンドアロン・インスタンスとして、すべて任意でミラーリングした状態で、Kubernetes プラットフォームに導入できます。

Kubernetes で InterSystems IRIS を導入するのに IKO は必須ではありませんが、プロセスが大幅に簡易化され、InterSystems IRIS 固有のクラスタ管理機能が Kubernetes に追加され、クラスタにノードを追加するなどのタスクが可能になります。このようなタスクは、IKO を使わなければインスタンスを直接操作して手動で行わなければなりません。

IKO の使用法の詳細は、"InterSystems Kubernetes Operator の使用" を参照してください。

2.2.1.2 構成マージを使用した分散キャッシュ・クラスタの導入

Linux および UNIX® システムで利用可能な構成マージ機能を使用すると、宣言型構成マージ・ファイルを導入内の各インスタンスに適用することにより、同じイメージから導入した InterSystems IRIS コンテナや、同じキットからインストールしたローカル・インスタンスの構成を変更することができます。

このマージ・ファイルは、既存のインスタンスを再起動したときに適用することもでき、インスタンスの構成パラメータ・ファイル (CPF) を更新します。CPF にはインスタンスのほとんどの構成設定が含まれており、これらの設定は開始時に毎回、インスタンス導入後の最初の設定を含めて CPF から読み取られます。導入時に構成マージを適用すると、インスタンスと共に提供された既定の CPF が実質的に独自の更新バージョンに置き換えられます。

構成マージを使用すると、データ・サーバとアプリケーション・サーバに対して別個のマージ・ファイルを呼び出して順番に導入することによって、分散キャッシュ・クラスタを導入できます。

前述のように、IKO には構成マージ機能が組み込まれています。

一般的な構成マージの使用法および具体的な分散キャッシュ・クラスタの導入方法の詳細は、"構成マージを使用した InterSystems IRIS の自動構成" を参照してください。

2.2.2 管理ポータルを使用したクラスタの導入

手動で導入する場合は、最初に、クラスタをホストするインフラストラクチャを特定またはプロビジョニングしてから、ホスト上で InterSystems IRIS インスタンスを特定または導入し、最後に、導入したインスタンスをクラスタとして構成する必要があります。追加する InterSystems IRIS インスタンスを導入または特定し、ホスト間に十分な帯域幅のネットワーク・アクセスを設定した後、管理ポータルを使用して分散キャッシュ・クラスタを構成するには、次の手順を実行します。

- データ・サーバの準備
- アプリケーション・サーバの構成

これらの手順は、管理ポータルの [ECP設定] ページで実行します ([**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**])。

2.2.2.1 データ・サーバの準備

InterSystems IRIS インスタンスは、アプリケーション・サーバ上で構成されるまで、分散キャッシュ・クラスタ内のデータ・サーバとして実際に動作させることはできません。ただし、インスタンスをデータ・サーバとして準備する手順には、1 つの必須アクションと 2 つのオプション・アクションが含まれています。

インスタンスをデータ・サーバとして準備するには、[ECP設定] ページに移動し (管理ポータルのホーム・ページで [**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**] の順に選択)、以下を実行します。

- 右側の [**このシステムをECPデータサーバとする**] ボックスで、ECP サービスの [**有効**] リンクをクリックすることにより、このサービスを有効にします。これにより %Service_ECP の [サービス編集] ダイアログが開きます。[**サービス有効**] を選択し、[**保存**] をクリックしてサービスを有効にします。(サービスが既に有効になっている場合 (ボックス内に [**無効**] リンクが存在することによって示されます)、次の手順に進みます)。

  注釈　InterSystems サービスの詳細な説明は、"サービス" を参照してください。

- 複数のアプリケーション・サーバを同時にデータ・サーバに接続できるようにする場合は、[**このシステムを ECP データサーバとする**] ボックスで、[**アプリケーションサーバの最大数**] 設定を、構成するアプリケーション・サーバの数に変更して、[**保存**] をクリックし、その後、インスタンスを再起動します(アプリケーション・サーバの同時接続の数が、この設定用に入力した数よりも大きくなると、データ・サーバ・インスタンスが自動的に再起動します)。

  注釈　アプリケーション・サーバの最大数は、UNIX® および Linux プラットフォームの構成マージ・ファイルに含まれる構成パラメータ・ファイル (CPF) の MaxServerConn 設定を使用して設定することもできます。

- [**トラブル状態の時間間隔**] 設定は、アプリケーション・サーバとデータ・サーバ間で中断された接続の回復を管理するために使用される 3 つのタイムアウトのうちのいずれかを決定します。長期的なクラスタの動作に関するデータを取得するまでは、既定の 60 のままとします。ECP リカバリ・タイムアウトの詳細は、"ECP リカバリ・プロトコル" を参照してください。

- TLS の使用を有効にしてアプリケーション・サーバからの接続をセキュリティ保護するには、[Set up SSL/TLS '%ECPServer'] リンクをクリックしてデータ・サーバの ECP TLS 構成を作成し、[**ECP SSL/TLS サポート**] 設定を次のように指定します。

  - [**必須**] – アプリケーション・サーバは、このデータ・サーバに対して [**SSL/TLS 使用**] が選択されている場合にのみ接続できます。
  - [**有効**] – アプリケーション・サーバは、このデータ・サーバに対して [**SSL/TLS 使用**] が選択されているかどうかに関係なく接続できます。
  - [**無効**] – アプリケーション・サーバは、このデータ・サーバに対して [**SSL/TLS 使用**] が選択されている場合 (既定)、接続できません。

  "分散キャッシュ・クラスタのセキュリティ" で説明しているとおり、TLS は ECP 通信をセキュリティ保護するためのいくつかのオプションのうちの 1 つです。ただし、TLS を有効にすると、パフォーマンスに重大な悪影響を及ぼす場合があります。クラスタのアプリケーション・サーバとデータ・サーバが同じデータ・センターに配置されることにより、最適なパフォーマンスが提供されている場合、データ・センター単独での物理的なセキュリティにより、クラスタに十分なセキュリティが提供される可能性があります。

  データ・サーバでのセキュリティで保護されたアプリケーション・サーバ接続の承認を含む、分散キャッシュ・クラスタでの TLS の有効化および使用に関する重要な情報は、"アプリケーション・サーバのデータ・サーバへの接続の TLS によるセキュリティ保護" を参照してください。

注釈　ECP は、データ・サーバ上のデータベース・キャッシュの一部を使用して、さまざまな制御構造を格納します。これに対応するために、データベース・キャッシュのサイズを増やすことが必要になる場合があります。詳細は、"ECP 制御構造用のデータ・サーバ・データベース・キャッシュの増大" を参照してください。

これで、データ・サーバは、有効なアプリケーション・サーバから接続できるようになります。

2.2.2.2 アプリケーション・サーバの構成

InterSystems IRIS インスタンスを分散キャッシュ・クラスタ内のアプリケーション・サーバとして構成するには、次の 2 つの手順が必要です。

- アプリケーション・サーバ・インスタンス上のデータ・サーバとして、データ・サーバ・インスタンスを追加します。
- アプリケーション・サーバ上のリモート・データベースとして、データ・サーバ上に目的のデータベースを追加します。

アプリケーション・サーバにデータ・サーバを追加するには、次の手順を実行します。

1. "データ・サーバの準備" のデータ・サーバに関する説明に従い、[ECP設定] ページに移動して、[**このシステムを ECPアプリケーションサーバとする**] 側の設定を既定のままとします。

2. 追加しているデータ・サーバの [ECP SSL/TLS サポート] 設定が [有効] または [必須] の場合は、[Set up SSL/TLS '%ECPClient'] リンクをクリックして、アプリケーション・サーバの ECP TLS 構成を作成します(これは、後で [ECPデータサーバ] ダイアログでも実行できます)。詳細は、次の手順の [SSL/TLS 使用] 設定を参照してください。

3. [データサーバ] をクリックして [ECP データサーバ] ページを表示し、[サーバ追加] をクリックします。[ECPデータサーバ] ダイアログに、データ・サーバについて次の情報を入力します。

   - **サーバ名** – データ・サーバを識別する説明的な名前(この名前は 64 文字に制限されます)。
   - **ホストDNS名またはIPアドレス** – データ・サーバのホストまたはその IP アドレスの DNS 名を指定します (ドットで区切った十進数形式。IPv6 を有効にしている場合は、コロン区切りの形式)。DNS 名で指定した場合、アプリケーション・サーバがそのデータ・サーバ・ホストに接続するたびに、実際の IP アドレスに解決されます。詳細は、"IPv6 のサポート" を参照してください。

     重要　　ミラー・プライマリをデータ・サーバとして追加する場合 ([ミラー接続] 設定を参照)、ミラーの仮想 IP アドレス (VIP) を入力するのではなく、現在のプライマリ・フェイルオーバー・メンバの DNS 名または IP アドレスを入力します。

   - [IP ポート] – このポート番号の既定は 1972 (既定の InterSystems IRIS スーパーサーバ (IP) ポート) です。必要に応じてデータ・サーバの InterSystems IRIS インスタンスのスーパーサーバ・ポートに変更します。
   - [ミラー接続] – データ・サーバがミラー内のプライマリ・フェイルオーバー・メンバである場合は、このチェック・ボックスにチェックを付けます (データ・サーバとしてのミラー・プライマリの構成に関する重要な情報は、"ミラーへのアプリケーション・サーバ接続の構成" を参照してください)。
   - [バッチモード] – このデータ・サーバのサーバ・プロセスをバッチ・モードで実行する必要がある場合はこのチェック・ボックスにチェックを付けます。バッチ・モードでは、データ・サーバは常にブロックをロードして、それらをバッチ・レベルでキャッシュします。ただし、ブロックがアプリケーション・サーバに戻されると、そのブロックは定期的にキャッシュされます。
   - [SSL/TLS 使用] – このチェック・ボックスは、以下のように使用します。
     - 追加しているデータ・サーバの [ECP SSL/TLS サポート] 設定が [無効] の場合、このチェック・ボックスにチェックを付けても付けなくても問題ありません。データ・サーバへの接続をセキュリティ保護するために TLS は使用されません。
     - 追加しているデータ・サーバの [ECP SSL/TLS サポート] 設定が [有効] のとき、このデータ・サーバへの接続をセキュリティ保護するのに TLS を使用する場合はこのチェック・ボックスにチェックを付け、TLS を使用しない場合はチェックを外します。
     - 追加しているデータ・サーバの [ECP SSL/TLS サポート] 設定が [必須] の場合は、このチェック・ボックスにチェックを付ける必要があります。

     追加しているデータ・サーバの [ECP SSL/TLS サポート] 設定が [有効] または [必須] で、アプリケーション・サーバに対してまだ TLS 構成を作成していない場合は、[Set up SSL/TLS '%ECPClient'] リンクをクリックしてこれを行います。データ・サーバでのセキュリティで保護されたアプリケーション・サーバ接続の承認を含む、分散キャッシュ・クラスタでの TLS の使用の詳細は、"アプリケーション・サーバのデータ・サーバへの接続の TLS によるセキュリティ保護" を参照してください。

4. [保存] をクリックします。データ・サーバはデータ・サーバ・リストに表示されます。利用可能なリンクを使用して、データ・サーバ定義を削除または編集したり、その状態を変更したりすることができます ("分散アプリケーションの監視" を参照)。データ・サーバ上の [ECP設定] ページに進み、[アプリケーションサーバ] ボタンをクリックすることで、データ・サーバへのすべてのアプリケーション・サーバ接続のリストを表示することもできます。

アプリケーション・サーバ上のリモート・データベースとしてデータ・サーバ上にそれぞれ必要なデータベースを追加するには、アプリケーション・サーバ上にネームスペースを作成し、次のようにそのデータベースにマッピングする必要があります。

1. 管理ポータルのホーム・ページで **[システム管理]**、**[構成]**、**[システム構成]**、**[ネームスペース]** の順に選択して、[ネームスペース] ページに移動します。**[新規ネームスペース作成]** をクリックして、[新規ネームスペース] ページを表示します。
2. 新規ネームスペースの名前を入力します。通常は、マッピングされるリモート・データベースの名前が反映されます。
3. **[このネームスペースにおけるグローバルの既定のデータベース]** で、**[リモートデータベース]**、**[新規データベース作成]** の順に選択して、[リモートデータベースを作成] ダイアログを開きます。このダイアログで、
   - **[リモートサーバ]** ドロップダウンからデータ・サーバを選択します。
   - **[リモートディレクトリ]** を **[リストからディレクトリを選択]** に設定し、データ・サーバ上のすべてのデータベース・ディレクトリを一覧表示する **[ディレクトリ]** ドロップダウンを使用して、ネームスペースにマッピングするデータ・サーバ・データベースを選択します。
   - リモート・データベースのローカル名を入力します。これは、通常、データ・サーバ上のデータベースの名前、前の手順で指定されたデータ・サーバのローカル名、またはその両方を反映します。
   - **[完了]** をクリックしてリモート・データベースを追加し、新しいネームスペースにマッピングします。
4. **[このネームスペースでルーチンのデフォルト・データベースは]** で、**[リモートデータベース]** を選択し、次に、ドロップダウンから作成したデータベースを選択します。
5. ネームスペースは相互運用対応である必要はありません。時間を節約するために、**[相互運用プロダクション用にネームスペースを有効化]** チェック・ボックスのチェックを外すことができます。
6. **[保存]** を選択します。[ネームスペース] リストに新規ネームスペースが一覧表示されます。

アプリケーション・サーバ上のリモート・データベースとしてデータ・サーバ・データベースを追加すると、アプリケーションは、アプリケーション・サーバにマッピングされているネーム・スペースを介してそのデータベースをクエリすることができます。

注釈　アプリケーション・サーバ上のネームスペースが、データ・サーバ上のデータベースにマッピングされていても、データ・サーバ上のそのデータベースにマッピングされているネームスペースへの変更は、アプリケーション・サーバにとっては不明であることに留意してください。(マッピングの詳細は、"グローバル・マッピング" を参照してください。)例えば、データ・サーバ上のネームスペース DATA に、既定のグローバル・データベース DATA があるとします。アプリケーション・サーバでは、REMOTEDATA というネームスペースは、同じ (リモート) データベース DATA にマッピングされます。データ・サーバ上の DATA ネームスペースでマッピングを作成して、グローバル ^DATA2 を DATA2 データベースにマッピングしても、このマッピングはアプリケーション・サーバに伝播されません。このため、アプリケーション・サーバ上のリモート・データベースとして DATA2 を追加しないで、REMOTEDATA ネームスペースに同じマッピングを作成しても、アプリケーション・サーバが受信するクエリでは、^DATA2 グローバルを読み取ることができません。

2.2.3 分散キャッシュ・クラスタのセキュリティ

分散キャッシュ・クラスタ内のすべての InterSystems インスタンスが、InterSystems IRIS の安全な境界の内部 (外部に対する安全が確保されている環境の内部) に存在する必要があります。その理由は、ECP が基本的なセキュリティ・サービスであり、リソース・ベースのサービスではないので、アクセスするユーザを制限する方法がないためです(基本サービスおよびリソース・ベースのサービスの詳細は、"使用可能なサービス" を参照してください)。

ただし、次のセキュリティ・ツールは利用可能です。

- アプリケーション・サーバのデータ・サーバへの接続の TLS によるセキュリティ保護
- データ・サーバへのアクセスの制限
- ロールと権限によるデータベースへのアクセスの制御

注釈　データ・サーバ上でデータベースが暗号化されている場合は、接続されているすべてのアプリケーション・サーバ上の IRISTEMP データベースも暗号化する必要があります。キーは、すべてのサーバで同じものにすることもできれば、別々のものにすることもできます。データベース暗号化の詳細は、"暗号化ガイド" を参照してください。

2.2.3.1 アプリケーション・サーバのデータ・サーバへの接続の TLS によるセキュリティ保護

TLS がデータ・サーバで有効な場合、これを使用してアプリケーション・サーバからそのデータ・サーバへの接続をセキュリティ保護できます。この保護には、X.509 証明書ベースの暗号化が含まれます。TLS およびインターシステムズ製品での TLS の使用の詳細は、インターシステムズの "TLS ガイド" を参照してください。

データ・サーバの構成または編集時、あるいはそれ以降であればいつでも ("データ・サーバの準備" を参照)、[ECP SSL/TLS サポート] 設定として、既定の [無効] ではなく、[有効] または [必須] を選択できます。これらの設定は、データ・サーバをアプリケーション・サーバに追加する ("アプリケーション・サーバの構成" を参照) か、既存のデータ・サーバを編集する際に、TLS によりデータ・サーバへの接続をセキュリティ保護する、[SSL/TLS 使用] チェック・ボックスの使用オプションを制御します。これら設定は、以下のように影響します。

- **無効** – アプリケーション・サーバのこのデータ・サーバへの接続に対する TLS の使用を無効にします。[SSL/TLS 使用] が選択されているアプリケーション・サーバであっても、同様です。
- **有効** – データ・サーバでアプリケーション・サーバの接続に対する TLS の使用が有効になります。[SSL/TLS 使用] が選択されているアプリケーション・サーバからの接続に対して TLS が使用され、[SSL/TLS 使用] が選択されていないアプリケーション・サーバからの接続に対しては、SSL/TLS は使用されません。
- **必須** – データ・サーバではアプリケーション・サーバの接続に TLS を使用する必要があります。アプリケーション・サーバは、データ・サーバに対して [SSL/TLS 使用] が選択されている場合のみ、データ・サーバに接続できます。この場合、すべての接続に対して TLS が使用されます。

TLS を使用してアプリケーション・サーバからデータ・サーバへの接続を確立するには、次の 3 つの要件があります。

- データ・サーバでは、スーパーサーバ・クライアントに対して TLS 接続を有効にする必要があります。そのためには、データ・サーバで、システムの既定のスーパーサーバの構成ページ ([**システム管理**]→[**セキュリティ**]→[**スーパーサーバ**]) に移動して、[SSL/TLS support level] 設定に [**有効**] を選択します。詳細は、"スーパーサーバの管理" を参照してください。
- 両方のインスタンスに ECP TLS 構成が必要です。

  このため、[ECP設定] ページ ([**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**]) の両側の [**このシステムをECPアプリケーションサーバとする**] と [**このシステムをECPデータサーバとする**] には、[SSL/TLS の構成] リンクが含まれており、これを使用してインスタンスに適した ECP TLS 構成を作成できます。このためには、以下の手順を実行します。

  1. [ECP設定] ページのアプリケーション・サーバ側の [Set up SSL/TLS '%ECPClient'] リンク、またはデータ・サーバ側の [Set up SSL/TLS '%ECPServer'] リンクをクリックします。
  2. [**ECP用 SSL/TLS 構成を編集**] ダイアログ内のフォームの各フィールドに入力します。これらは [**新規 SSL/TLS 構成**] ページに似ています ("TLS 構成の作成または編集" を参照)。ただし、[**構成名**]、[**説明**]、[**有効**] のいずれのフィールドもありません。また、秘密鍵パスワードに関しては、このページでパスワードの入力または置き換え ([**新規パスワード入力**])、パスワードを使用しないことの指定 ([**パスワードクリア**])、あるいは既存のパスワードをそのまま使用すること ([**そのままにする**]) の指定を行えます。

     このページのフィールドは以下のとおりです。

     - [**信頼された証明書機関 X.509 証明書を含むファイル**]

       この構成が信頼する 1 つまたは複数の認証機関 (CA) の X.509 証明書 (PEM 形式) が含まれているファイルのパスと名前。絶対パス、または install-dir/mgr/ ディレクトリを基準とした相対パスで指定できます。

X.509 証明書とそれらの生成および使用の詳細は、インターシステムズの "TLS ガイド" を参照してください。

注釈　このファイルには、他のミラー・メンバに属する X.509 証明書の検証に使用できる証明書が含まれている必要があります。ファイルに複数の証明書が含まれている場合は、それらの証明書を正しい順序で(現在のインスタンスの証明書が最初になるように)並べる必要があります。詳細は、"必須証明書チェーンの確立" を参照してください。

- **[この構成のX.509 証明書を含むファイル]**

  構成独自の X.509 証明書 (PEM 形式) の場所。これは、絶対パスと相対パスのいずれかとして指定できます。

  注釈　証明書の識別名 (DN) は証明書のサブジェクト・フィールドに表示する必要があります。

- **[関連づけられた秘密鍵を含むファイル:]**

  構成の秘密鍵ファイルを格納する場所。絶対パスまたは相対パスで指定します。

- **[秘密鍵タイプ]**

  秘密鍵の生成に使用するアルゴリズム。有効なオプションは **[DSA]** と **[RSA]** です。

- **[パスワード]**

  ECP TLS 構成を作成している場合、**[新規パスワード入力]** を選択すると、証明書に関連付けられた秘密鍵のパスワードの入力および確認のための再入力を実行できます。

- **[プロトコル]**

  構成で有効と見なされる通信プロトコル TLSv1.1 および TLSv1.2 は、既定で有効となります。

- **[有効な暗号スイート]**

  クライアントとサーバ間の通信の保護に使用される暗号スイート・セット。通常、これは既定の設定のままにできます。

フォームの入力が完了したら **[保存]** をクリックします。

- アプリケーション・サーバは、TLS を使用して接続する前に、データ・サーバで承認する必要があります。

  アプリケーション・サーバが最初に TLS を使用してデータ・サーバに接続を試みると、その SSL (TLS) コンピュータ名 (その X.509 証明書の所有者識別名) とそのホストの IP アドレスが、データ・サーバの [アプリケーションサーバ] ページ (**[システム管理]→[構成]→[接続性]→[ECP設定]→[アプリケーションサーバ]**) の、**承認または拒否が決まっていない保留中の ECP アプリケーション・サーバ**のリストに表示されます。**[承認]** および **[拒否]** リンクを使用して、リスト内の要求に対応します(保留中の要求がない場合、リストは表示されません)。

  TLS を使用して接続することが承認されているアプリケーション・サーバが 1 つ以上存在する場合、その SSL (TLS) コンピュータ名が、[アプリケーションサーバ] ページの、**ECP アプリケーション・サーバとして承認された SSL コンピュータ名**のリストに表示されます。承認をキャンセルするには **[削除]** リンクを使用します(承認されたアプリケーション・サーバがない場合、リストは表示されません)。

2.2.3.2 データ・サーバへのアクセスの制限

既定では、(前のセクションで説明したように) データ・サーバ・インスタンスがデータ・サーバとして構成されている InterSystems IRIS インスタンスは、データ・サーバに接続できます。ただし、許可される着信接続元となるホストを指定することによって、データ・サーバに対するアプリケーション・サーバとして機能できるインスタンスを制限できます。これを行う場合、明示的にリストされていないホストは、データ・サーバに接続できません。これを行うには、データ・サーバ上で次の手順を実行します。

1. [サービス] ページ (ポータルのホーム・ページから、**[セキュリティ]**、**[サービス]** の順に選択) で、[%Service_ECP] をクリックします。[サービス編集] ダイアログが表示されます。
2. 既定では、**[許可済みの接続元]** ボックスは空です。つまり、ECP サービスが有効な場合、すべてのアプリケーション・サーバがこのインスタンスに接続できます。**[追加]** をクリックして、単一の IP アドレス (192.9.202.55 など) または完全修飾ドメイン名 (mycomputer.myorg.com など)、あるいは IP アドレスの範囲 (例えば、8.61.202–210.* や 18.68.*.*) を入力します。リストに 1 つ以上のエントリがあるときに、[サービス編集] ダイアログで **[保存]** をクリックすると、これらのエントリで指定されたホストのみが接続できます。

リストには説明に従っていつでもアクセスすることができます。リストからホストを削除するには **[削除]** を使用し、ホストに関連付けられたロールを指定するには **[編集]** リンクを使用します ("ロールと権限によるアクセスの制御" を参照)。

2.2.3.3 ロールと権限によるデータベースへのアクセスの制御

インターシステムズが使用するセキュリティ・モデルでは、データベースを含むアセットがリソースに割り当てられ、リソースには読み取りや書き込みなどの許可が割り当てられます。リソースと許可の組み合わせを権限と呼びます。権限は、ユーザが所属できるロールに割り当てられます。このように、リソースへのユーザ・アクセスを制御するためにロールが使用されます。このモデルの詳細は、"承認 : ユーザ・アクセスの制御" を参照してください。

データ・サーバにあるデータベースへのアクセス許可を取得するには、アプリケーション・サーバ上のプロセスを開始するユーザが保持するロールと、データ・サーバ上の ECP 接続に対して設定されたロールの両方に、そのデータベースを表す同じリソースに対する許可が指定されている必要があります。例えば、あるユーザが、特定のデータベース・リソースに対する読み取り許可の権限を付与するアプリケーション・サーバ上のロールに所属し、データ・サーバ上の ECP 接続に設定されているロールにもこの権限が含まれている場合、そのユーザは、アプリケーション・サーバ上のデータベースからデータを読み取ることができます。

アプリケーション・サーバの代わりにデータ・サーバを実行する場合、既定では、InterSystems IRIS は、データ・サーバ上の ECP 接続に **%All** 特権を与えます。つまり、アプリケーション・サーバ上のユーザが持つ権限はすべて、データ・サーバ上で照合され、アクセスはアプリケーション・サーバ上でのみ制御されます。例えば、**%DB_USER** リソースに対する特権のみを持っており、**%DB_IRISLIB** リソースに対する特権を持っていない、アプリケーション・サーバのユーザは、データ・サーバの USER データベースにあるデータにアクセスできますが、データ・サーバの IRISLIB データベースにアクセスしようとすると、<PROTECT> エラーとなります。アプリケーション・サーバの別のユーザが、**%DB_IRISLIB** リソースに対する特権を持っている場合、そのユーザは IRISLIB データベースを利用できます。

注釈　LDAP サーバを使用して、分散キャッシュ・クラスタのアプリケーション・サーバ全体で、ユーザ・ロールと特権を含む、一元的なセキュリティを実装することをお勧めします。InterSystems IRIS での LDAP の使用についての詳細は、"LADP ガイド" を参照してください。

ただし、アプリケーション・サーバのホストに基づいて、データ・サーバ上の ECP 接続で使用できるロールを制限することもできます。例えば、データ・サーバで、特定のアプリケーション・サーバと対話するときに使用できるロールを **%DB_USER** のみとすることを指定できます。この場合、アプリケーション・サーバで **%DB_USER** ロールが与えられているユーザは、データ・サーバの USER データベースにアクセスできますが、アプリケーション・サーバ上のユーザは、付与されたロールに関係なく、データ・サーバ上の他のデータベースにアクセスすることはできません。

注意　データ・サーバですべての ECP 接続に対して **%All** 権限を引き続き付与できるようにするのではなく、クラスタ内のすべてのアプリケーション・サーバで使用可能なロールを指定することにより、クラスタを保護することを強くお勧めします。

以下は、この動作に対する例外です。

- InterSystems IRIS は必ずデータ・サーバに **%DB_IRISSYS** ロールを与えます。これは、このデータ・サーバが稼動するには、IRISSYS データベースへの Read アクセスが必要なためです。つまり、アプリケーション・サーバ上で **%DB_IRISSYS** ロールを持つユーザは、データ・サーバの IRISSYS データベースにアクセスできます。

このアプリケーション・サーバのユーザが、データ・サーバの IRISSYS データベースにアクセスできないようにする方法には次の 2 とおりがあります。

- **%DB_IRISSYS** リソースに対する特権をユーザに与えない。
- データ・サーバで、IRISSYS データベースのリソース名を、**%DB_IRISSYS** 以外の名前に変更する。このとき、アプリケーション・サーバのユーザが変更後の名前を持つリソースに対する特権を持っていないことを確認してください。

- データ・サーバにパブリック・リソースが存在する場合、アプリケーション・サーバのロールや、ECP 接続用に構成されたロールに関係なく、ECP アプリケーション・サーバ上のすべてのユーザはこのパブリック・リソースを使用できます。

データ・サーバ上の特定のアプリケーション・サーバからの ECP 接続に使用可能なロールを指定するには、以下の操作を行います。

1. [サービス] ページ (ポータルのホーム・ページから、**[セキュリティ]**、**[サービス]** の順に選択) に移動して、[%Service_ECP] をクリックし、[サービス編集] ダイアログを表示します。
2. 制限するアプリケーション・サーバ・ホストの **[編集]** リンクをクリックして、**[ロールの選択]** 領域を表示します。
3. ホストのロールを指定するには、**[使用可能]** の下に一覧表示されるロールから必要なロールを選択し、右矢印をクリックして、選択したロールを **[選択済み]** リストに追加します。
4. **[選択済み]** リストからロールを削除するには、削除するロールを選択し、左矢印をクリックします。
5. **[選択済み]** リストにすべてのロールを追加するには、二重右矢印をクリックします。また、**[選択済み]** リストからロールをすべて削除するには、二重左矢印をクリックします。
6. **[保存]** をクリックして、IP アドレスとロールを関連付けます。

既定では、リストされたホストは **%All** ロールを保持していますが、その他のロールを 1 つまたは複数指定すると、この接続は指定したそれらのロールのみを持つことになります。したがって、**%Operator** ロールを持つホストまたは IP 範囲からの接続が持つ特権は、そのロールに関連付けられているもののみです。一方、ロールが関連付けられていない (つまり、**%All** ロール) ホストからの接続はすべての特権を持つことになります。

アプリケーション・サーバのホストで使用可能なロールに対する変更、およびデータ・サーバのリソース上のパブリック許可に対する変更を有効にするには、InterSystems IRIS の再起動が必要です。

2.2.3.4 セキュリティ関連のエラー報告

ECP で発生するセキュリティ関連のエラー報告の動作は、アプリケーション・サーバとデータ・サーバのどちらでチェックが失敗したか、またどのような操作が行われたかによって次のように異なります。

- アプリケーション・サーバでチェックに失敗した場合は、直ちに <PROTECT> エラーとなります。
- データ・サーバでの同期操作の場合は、直ちに <PROTECT> エラーとなります。
- データ・サーバでの非同期操作の場合は、<NETWORK DATA UPDATE FAILED> エラーが遅延時間の後で表示されることがあります。このような操作には、例えば Set 操作があります。

2.3 分散キャッシュ・アプリケーションの監視

実行中の分散キャッシュ・クラスタは、1 つ以上のアプリケーション・サーバ・システム (データ・コンシューマ) に接続されたデータ・サーバ・インスタンス (データ・プロバイダ) で構成されます。各アプリケーション・サーバとデータ・サーバ間では、ECP 接続が確立されています。つまり、ECP がデータとコマンドの送信に使用する TCP/IP 接続です。

[ECP設定] ページ ([**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**]) で、分散キャッシュ・クラスタ内のサーバと接続の状態を監視できます。

[ECP設定] ページには、次の 2 つのサブセクションがあります。

1. **[このシステムをECPデータサーバとする]** では、データ・サーバの設定と、ECP サービスの状態が表示されます。
2. **[このシステムをECPアプリケーションサーバとする]** には、アプリケーション・サーバの設定が表示されます。

以下のセクションでは、接続状態の情報について説明します。

- ECP 接続情報
- ECP 接続の状態
- ECP 接続処理

2.3.1 ECP 接続情報

[ECPデータサーバの設定] ページ ([**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**]) で **[データサーバ]** ボタンをクリックすると、アプリケーション・サーバ上の現在のデータ・サーバ接続を一覧表示する [ECPデータサーバ] ページが表示されます。[ECP設定] ページの **[アプリケーションサーバ]** ボタンをクリックすると表示される [ECPアプリケーションサーバ] ページには、データ・サーバ上の現在のアプリケーション・サーバ接続のリストが表示されます。

2.3.1.1 データ・サーバ接続

[ECPデータサーバ] ページに、各データ・サーバ接続について次の情報が表示されます。

サーバ名

アプリケーション・サーバ構成にサーバが追加されたときに入力された、この接続のデータ・サーバ・システムの論理名。

ホスト名

アプリケーション・サーバ構成にサーバが追加されたときに入力された、データ・サーバ・システムのホスト名。

IPポート

データ・サーバへの接続に使用する IP ポート番号。

ステータス

現在の接続の状態接続状態については、"ECP 接続の状態" セクションで説明します。

編集

この接続の現在の状態が**未接続**または**無効**である場合は、データ・サーバのポートおよびホスト情報を編集できます。

状態の変更

各データ・サーバ行から、そのデータ・サーバとの間に存在するECP接続の状態を変更できます。詳細は、"ECP 接続処理" のセクションを参照してください。

削除

アプリケーション・サーバからデータ・サーバの情報を削除できます。

2.3.1.2 アプリケーション・サーバ接続

[ECP設定]ページ ([**システム管理**]→[**構成**]→[**接続性**]→[**ECP設定**]) で [**ECP アプリケーションサーバ**] をクリックして、このデータ・サーバ上のアプリケーション・サーバ接続のリストを含む [ECP アプリケーションサーバ] ページを表示します。

クライアント名

この接続上のアプリケーション・サーバの論理名。

ステータス

現在の接続の状態接続状態については、"ECP 接続の状態" セクションで説明します。

クライアントIP

アプリケーション・サーバのホスト名または IP アドレス

IPポート

アプリケーション・サーバへの接続に使用するポート番号。

2.3.2 ECP 接続の状態

稼働中のクラスタでは、ECP 接続は以下のいずれかの状態になります。

テーブル 2-1: ECP 接続の状態

状態	説明
未接続	接続は定義済みですが、未使用です。
接続中	接続中です。接続されるまで表示される一時的な状態です。
正常	接続は正常に動作し、現在使用されています。
障害	接続に障害が発生しました。可能な場合、接続は自動的に修正されます。
無効	接続は、システム管理者によって手動で無効にされました。接続中のアプリケーションは <NETWORK> エラーを受け取ります。

以下のセクションでは、アプリケーション・サーバとデータ・サーバの接続状態を説明します。

- アプリケーション・サーバの接続状態
- データ・サーバの接続状態

2.3.2.1 アプリケーション・サーバの接続状態

以下のエントリは、アプリケーション・サーバ側のそれぞれの接続状態を示します。提示される数値によって、ログ・メッセージに示された接続状態を判断できます。例えば、以下のメッセージは、 : jojo96HABER を表します。

```
01/28/24-00:00:11:859 (6552) 2
[SYSTEM MONITOR]
ECPClientState Alert: ECP reports Clients state 6
```

- 初期化されていない (0)
- 未接続 (1)

- 接続中 (2)
- 接続失敗 (3)
- 無効 (4)
- 正常 (5)
- 障害 (6)
- リカバリ中 (7)

アプリケーション・サーバは初期化されていない (0)

ノードは、初期化中の状態 (きわめてまれ) か、まだ初期化されていません。

アプリケーション・サーバの未接続状態 (1)

アプリケーション・サーバ側の ECP 接続は、最初は未接続状態になっています。この状態の場合、接続のための ECP デーモンは存在しません。アプリケーション・サーバ・プロセスがネットワーク要求を送信すると、接続のためのデーモンが作成され、接続は接続中状態に入ります。

アプリケーション・サーバの接続中状態 (2)

接続中状態では、接続のためのネットワーク・デーモンが存在し、データ・サーバへの接続の確立を試行します。接続が確立されたら、正常状態に入ります。接続が接続中状態にある場合、ユーザ・プロセスは、接続が確立されるまで最大 20 秒待機します。接続がその時間内に確立されなかった場合、ユーザ・プロセスは <NETWORK> エラーを受け取ります。

アプリケーション・サーバ ECP デーモンは、バックグラウンドでデータ・サーバへの新しい接続を試行します。接続が 20 分以内に確立されなかった場合は、接続は未接続状態に戻り、接続のデーモンは削除されます。

アプリケーション・サーバの接続失敗状態 (3)

接続中の状態で行われた接続試行が失敗しました。この状態が数秒間続いた後、アプリケーション・サーバの未接続状態に移行します。

アプリケーション・サーバの無効状態 (4)

ECP 接続が無効としてマークされるのは、管理者が無効にすると判断した場合です。この状態では、デーモンは存在せず、その接続を使用するすべてのネットワーク要求はすぐに <NETWORK> エラーを受け取ります。

アプリケーション・サーバの正常状態 (5)

接続が完了した後、正常な (データ転送) 状態に入ります。この状態では、アプリケーション・サーバ側のデーモンが存在し、ネットワークを介して要求を送信し、応答を受信します。接続の正常状態は、接続が継続不可能になるか、アプリケーション・サーバかデータ・サーバから接続のシャットダウンが要求されるまで維持されます。

アプリケーション・サーバの障害状態 (6)

アプリケーション・サーバからデータ・サーバへの接続で問題が発生した場合、アプリケーション・サーバの ECP 接続は、障害状態に入ります。この状態では、アプリケーション・サーバの ECP デーモンが存在し、接続のリストアを試行します。基本の TCP 接続は、確立されている場合も確立されていない場合もあります。基本の TCP 接続がリセットされ、再現の必要があるかどうかによってリカバリ方法が異なることはありません。つまり、TCP 接続が一時的に機能を停止した場合でもリカバリ方法は同じです。

アプリケーション・サーバの **[リカバリまでの待機時間]** のタイムアウト期間中 (既定は 20 分)、アプリケーション・サーバは ECP 接続をリカバリするために、データ・サーバへの再接続を試行します。この間、既存のネットワーク要求は保持されますが、アプリケーション・サーバ側のユーザ・プロセスは、新しいネットワーク要求の送信を中止して、接続が再開されるのを待ちます。接続が **[リカバリまでの待機時間]** のタイムアウト期間内に回復した場合は、正常状態に戻り、中止されていたネットワーク要求は送信されます。

例えば、データ・サーバがオフラインになった場合、データ・サーバが使用可能になるまで、接続中のすべてのアプリケーション・サーバの状態は障害に設定されます。障害が適切に修正されると、接続状態は正常に戻ります。一方、障害状態をリカバリできない場合、未接続となります。

アプリケーションは、ネットワーク・アクセスが必要となるまで、継続して稼動します。サーバが応答していない間は、ローカルにキャッシュされたデータすべてをアプリケーションで使用できます。

アプリケーション・サーバのリカバリ中状態 (7)

リカバリ中状態は、障害状態の一部です。現在データ・サーバに TCP 接続がなく、新しい接続が確立された場合、アプリケーション・サーバとデータ・サーバはリカバリ・プロトコルに従って、アプリケーション・サーバのキャッシュをフラッシュし、トランザクションおよびロックを回復し、正常状態に戻ります。

同様に、データ・サーバのシャットダウンや、クラッシュの結果としてのシャットダウン後の再起動の場合、その後アプリケーション・サーバには再接続と既存のセッションを回復するための短い時間 (約 30 秒) が与えられます。そしてここでも、アプリケーション・サーバとデータ・サーバはリカバリ・プロトコルに従います。

接続のリカバリが、**[リカバリまでの待機時間]** のタイムアウト期間内に完了しなかった場合、アプリケーション・サーバは接続のリカバリを終了します。具体的には、アプリケーション・サーバは、すべての未実行のネットワーク要求にエラーを返し、接続状態を未接続に変更します。この作業がアプリケーション・サーバ側で行われなかった場合は、次にこのアプリケーション・サーバからデータ・サーバに接続されたときに、すべてのトランザクションはロールバックされ、このアプリケーション・サーバからのすべてのロックが解除されます。

リカバリに成功した場合、接続は正常状態に戻り、中止されていたネットワーク要求が送信されます。

2.3.2.2 データ・サーバの接続状態

以下のセクションでは、データ・サーバ側の各接続状態について説明します。

- 解放
- 正常
- 障害
- リカバリ

データ・サーバの解放状態

ECP サーバ・インスタンスが起動されたとき、ECP サーバに対するすべての接続は、初期の“割り当てのない”解放の状態にあり、接続アクセス・コントロール・リストにあるすべてのアプリケーション・サーバから接続できる状態になっています。アプリケーション・サーバからの接続が既にあり、その後切断されたが、リカバリ手順を行う必要がない場合、その接続は、“アイドル”状態である解放状態になります。この 2 つの状態の唯一の違いは、アイドル状態の場合は、その接続ブロックが既に特定のアプリケーション・サーバに割り当てられており、アクセス・コントロール・リストにある他のアプリケーション・サーバでは利用できない点です。

データ・サーバの正常状態

データ・サーバの正常状態では、アプリケーション・サーバの接続は正常です。接続を受け入れる側の処理ではどの時点でも、アプリケーション・サーバがデータ・サーバとの接続を切断した場合 (データ・サーバ自体のシャットダウン・シーケンスの場合は除く)、データ・サーバは未実行のトランザクションをロールバックし、そのアプリケーション・サーバからのロックを解除し、そのアプリケーション・サーバとの接続を“アイドル”状態である解放状態にします。

データ・サーバの障害状態

アプリケーション・サーバが応答していない場合、データ・サーバは、障害状態を示します。データ・サーバがクラッシュまたはシャットダウンした場合、サーバはクラッシュやシャットダウン時点でアクティブであった接続を記憶しています。再起動後、データ・サーバは、アプリケーション・サーバがセッション（ロックおよび開いているトランザクション）を再要求するまで、短時間待機します（通常は 30 秒）。このリカバリ待機時間中に、アプリケーション・サーバが完全にリカバリできなかった場合、その接続にある未実行の作業はすべてロールバックされ、接続は“アイドル”状態に置かれます。

データ・サーバのリカバリ状態

データ・サーバの接続は、アプリケーション・サーバがそのセッションの再要求プロセスにある間、ごく短時間、リカバリ状態となります。データ・サーバは、接続を再要求できるようにアプリケーション・サーバを **[トラブル状態の時間間隔]** のタイムアウト期間の間（既定値は 60 秒）障害状態に保ちます。接続が再要求されない場合、データ・サーバはアプリケーション・リソースを解放（開いているトランザクションすべてをロールバックし、ロックを解放）し、状態を解放に設定します。

2.3.3 ECP 接続処理

アプリケーション・サーバ上の［ECPデータサーバ］ページ（**[システム管理]→[構成]→[接続性]→[ECP設定]** の順に選択して、**[データサーバ]** ボタンをクリック）で、ECP 接続のステータスを変更できます。各データ・サーバの行で、**[状態の変更]** をクリックし、接続情報を表示して、以下の中から適切な操作を実行します。

「無効」に変更

接続の状態を 無効 に設定これにより、アプリケーション・サーバに設定されているロックを解放し、接続に関連するオープン・トランザクションをロールバックし、データ・サーバからキャッシュされたブロックを消去します。これがアクティブな接続である場合、状態の変更によって、データ・サーバからのネットワーク応答を待機しているすべてのアプリケーションにエラーが送信されます。

「正常」に変更

接続の状態を「正常」に設定

「未接続」に変更

接続の状態を「未接続」に設定状態を **[無効]** に変更する場合と同様に、これにより、アプリケーション・サーバに設定されているロックが解放され、接続に関連するオープン・トランザクションがロールバックされ、データ・サーバからキャッシュされたブロックが消去されます。これがアクティブな接続である場合、状態の変更によって、データ・サーバからのネットワーク応答を待機しているすべてのアプリケーションにエラーが送信されます。

2.4 分散キャッシュ・アプリケーションの開発

ここでは、アプリケーションの開発と設計の問題について説明します。オプションあるいは主要構成のいずれかとして分散キャッシュ・クラスタ上でアプリケーションを展開する場合は、このトピックを参照してください。

InterSystems IRIS を使用し、分散システムとしてアプリケーションを展開することは、基本的に実行時の構成に関係します（"分散キャッシュ・クラスタの導入" を参照してください）。つまり、InterSystems IRIS 構成ツールを使用し、データの論理名（グローバル）とアプリケーション・ロジック（ルーチン）を適切なシステムの物理ストレージにマップします。

このセクションでは、以下のトピックについて説明します。

- ECP リカバリ・プロトコル
- 切断の強制

- パフォーマンスとプログラミングに関する考慮点
- ECP 関連エラー

2.4.1 ECP リカバリ・プロトコル

ECP は、アプリケーション・サーバとデータ・サーバの間の接続の中断から自動的に回復するように設計されています。このような中断が発生した場合、ECP はリカバリ・プロトコルを実行します。リカバリ・プロトコルは、障害の性質によって異なります。その結果は、接続が回復されるか、何も発生しなかったかのようにアプリケーションのプロセスを続行できるか、またはリセットされ、トランザクションを強制的にロールバックし、アプリケーションのプロセスが再構築されます。主な原則は以下のとおりです。

- アプリケーション・サーバとデータ・サーバの間の接続が中断した場合、アプリケーション・サーバはデータ・サーバとの接続を再確立しようとします。必要に応じて、**[再接続までの時間]** 設定で指定された間隔 (既定値は 5 秒) で繰り返し実行します。
- 中断が短時間の場合、接続は回復されます。

  データサーバで構成された **[トラブル状態の時間間隔]** のタイムアウト期間 (既定値は 60 秒) 内に接続が再確立された場合、すべてのロックおよび開いているトランザクションが中断前の状態にリストアされます。
- 中断が長くなると、データ・サーバは接続をリセットします。このため、中断が終了したときに回復できなくなります。

  [トラブル状態の時間間隔] 内に接続が再確立されない場合、データ・サーバは接続を一方的にリセットします。これにより、トランザクションをロールバックして、応答しないアプリケーション・サーバからのロックを解放できるようになり、機能しているアプリケーション・サーバがブロックされなくなります。接続が回復した場合、アプリケーション・サーバの側からすると接続が無効になります。中断された接続でデータ・サーバを待機しているすべてのプロセスは、<NETWORK> エラーを受け取り、rollback-only 条件になります。アプリケーション・サーバによって受信された次の要求では、データ・サーバへの新しい接続が確立されます。
- 中断が非常に長い場合は、アプリケーション・サーバも接続をリセットします。

  アプリケーション・サーバの長い Time to wait for recovery タイムアウト期間 (既定は 20 分) 内に接続が再確立されない場合、アプリケーション・サーバは一方的に接続をリセットします。中断された接続でデータ・サーバを待機しているすべてのプロセスは、<NETWORK> エラーを受け取り、rollback-only 条件になります。アプリケーション・サーバによって受信された次の要求では、データ・サーバへの新しい接続が確立されます (可能な場合)。

ECP タイムアウト設定を、以下の表に示します。各項目は、管理ポータルの **[システム]→[構成]→[ECP設定]** ページ、または構成パラメータ・ファイル (CPF) の ECP セクションで構成できます (詳細は、"構成パラメータ・ファイル・リファレンス" の "ECP" を参照)。

テーブル 2-2: ECP のタイムアウト値

管理ポータルの設定	CPF 設定	既定値	範囲	
再接続までの時間	ClientReconnectInterval	5 秒	1 ～ 60 秒	アプリケーションがデータ・サーバに再接続しようとする間隔。
トラブル状態の時間間隔	ServerTroubleDuration	60 秒	20 ～ 65535 秒	中断された接続をリセットする前に、データ・サーバがアプリケーション・サーバからの通信を待機する時間。
リカバリまでの待機時間	ClientReconnectDuration	1200 秒 (20分)	10 ～ 65535 秒	中断された接続をリセットする前に、アプリケーション・サーバがデータ・サーバへの再接続の試行を続ける時間。

既定値は、以下を実現することを目的としています。

- アプリケーション・サーバが使用可能になるのを待機することによって、他のアプリケーション・サーバで使用できるデータ・サーバ・リソースを長時間拘束しないようにする。
- データ・サーバが利用できないときに他に何もすることがないアプリケーション・サーバに、頻繁に再接続を試みることにより、長時間の接続中断を待つ機能を提供する。

ECP は、TCP の物理接続を利用して、容量をあまり使用せずに、相手側のインスタンスの正常性を検出します。ほとんどのプラットフォームで、システム・レベルで TCP 接続障害および検知動作を調整できます。

アプリケーション・サーバの接続が非アクティブになると、データ・サーバは、その接続で新しい要求が到着するのを待機するか、またはアプリケーション・サーバによって新しい接続が要求されるのを待機しながら、アクティブなデーモンを維持します。既存の接続から要求が着信した場合、リカバリすることなく、すぐに処理が再開されます。基本のハートビート・メカニズムが、システムまたはネットワークの障害のためにアプリケーション・サーバが完全に利用不可であることを示す場合、基本の TCP 接続は迅速にリセットされます。したがって、アプリケーション・サーバから長時間応答がないことは、一般に、アプリケーション・サーバの機能を停止させるがその接続を妨げることのない、何らかの問題を示します。

基本の TCP 接続がリセットされた場合、データ・サーバはその接続を“再接続の待機”状態にします。この状態では、データ・サーバ側にアクティブな ECP デーモンは存在しません。アプリケーション・サーバから新しい接続が要求されたときに、1 対のデータ・サーバ・デーモンが作成されます。

この非応答状態と再接続の待機状態を総称して、データ・サーバの障害状態と呼びます。どちらの状態でも必要なリカバリは、ほとんど同じです。

データ・サーバで障害またはシャットダウンが発生した場合、サーバはクラッシュやシャットダウン時点でアクティブであった接続を記憶しています。再起動後、データ・サーバは短時間 (通常は 30 秒) の間、中断した接続を再接続の待機状態に置きます。その状態の間に、アプリケーション・サーバとデータ・サーバは連携して、データ・サーバがシャットダウンした時点以降のすべてのトランザクションとロック、および未実行の Set および Kill トランザクションを回復します。

ECP 構成インスタンスのリカバリ中、InterSystems IRIS は多数のリカバリ・メカニズムを保証し、またこれらの保証に対する制限を指定します。″ECP リカバリ・プロセス、保証、および制限″では、これらの詳細と、リカバリ・プロセスの詳細について説明しています。

2.4.2 切断の強制

既定では、ECP は、アプリケーション・サーバとデータ・サーバ間の接続を自動的に管理します。ECP 構成インスタンスが起動した時点では、アプリケーション・サーバとデータ・サーバ間のすべての接続は、未接続 (接続が定義済みでも、確立されていない) 状態になります。アプリケーション・サーバが、データ・サーバへの接続を必要とする (データやコードに対する) 要求を行うと、その接続は自動的に確立され、接続状態は正常に変更されます。アプリケーション・サーバとデータ・サーバ間のネットワーク接続は、その後、接続された状態を維持します。

アプリケーションの中には、ECP 接続を切断する必要があるものものあります。例えば、アプリケーション・サーバとして構成されたシステムがあるとします。このシステムは、データ・サーバ・システムに格納されたデータを定期的に (1 日に数回) フェッチする必要がありますが、その後は、データ・サーバとのネットワーク接続をオープンにしておく必要がありません。この場合、アプリケーション・サーバ・システムは、SYS.ECP.ChangeToNotConnected() メソッドを呼び出し、ECP 接続状態を強制的に未接続にします。

以下に例を示します。

ObjectScript

```
Do OperationThatUsesECP()
Do SYS.ECP.ChangeToNotConnected("ConnectionName")
```

ChangeToNotConnected メソッドは、以下を実行します。

1. 変更したデータをデータ・サーバに送信し、データ・サーバからの確認応答を待機します。
2. アプリケーション・サーバがオープンしたデータ・サーバ上のロックを削除します。

3. データ・サーバ側のオープンしているすべてのトランザクションをロールバックします。アプリケーション・サーバ側のトランザクションは、“rollback only”状態になります。
4. 未実行の要求を NETWORK エラーで完了します。
5. キャッシュされているすべてのブロックをフラッシュします。

状態が未接続に変更された後、次にデータ・サーバにデータが要求されると、接続は自動的に再確立されます。

注釈　管理ポータルからのデータ・サーバ接続ステータスの変更については、"データ・サーバ接続" を参照してください。

2.4.3 パフォーマンスとプログラミングに関する考慮点

分散キャッシュ・クラスタ・ベースのアプリケーションの高性能性および高信頼性を実現するには、以下の問題を認識する必要があります。

- 複数の ECP チャネルを使用しない
- ECP 制御構造用のデータ・サーバ・データベース・キャッシュの増大
- 負荷分散のユーザ要求の影響の評価
- トランザクションを単一のデータ・サーバに制限する
- アプリケーション・サーバでの一時グローバルの配置
- 未定義のグローバルに対する繰り返し参照の回避
- ストリーム・フィールドの使用の回避
- $Increment 関数とアプリケーション・カウンタ

2.4.3.1 複数の ECP チャネルを使用しない

帯域幅を増やすためにアプリケーション・サーバとデータ・サーバとの間に複数の重複した ECP チャネルを確立しないでください。1 つの論理トランザクションに対するロックや更新がデータ・サーバ上で非同期になり、データが不整合になるリスクがあります。

2.4.3.2 ECP 制御構造用のデータ・サーバ・データベース・キャッシュの増大

ECP 経由で提供されるブロックをバッファリングする処理に加え、データ・サーバでは、さまざまな ECP 制御構造を格納するのにグローバル・バッファを使用します。この構造に必要なメモリ量を決定する要素はいくつかありますが、最も重要なものは、クライアントのキャッシュの集約サイズの関数です。要求サイズを概算するために、必要に応じてデータ・サーバのデータベース・キャッシュを調整できます。以下のガイドラインを使用してください。

データベースのブロック・サイズ	推奨事項
8 KB	50 MB にアプリケーション・サーバのすべての 8 KB データベース・キャッシュのサイズの合計の 1% を加えたサイズ
16 KB (有効化されている場合)	アプリケーション・サーバのすべての 16 KB データベース・キャッシュのサイズの合計の 0.5%
32 KB (有効化されている場合)	アプリケーション・サーバのすべての 32 KB データベース・キャッシュのサイズの合計の 0.25%
64 KB (有効化されている場合)	アプリケーション・サーバのすべての 64 KB データベース・キャッシュのサイズの合計の 0.125%

例えば、既定の 8 KB のブロック・サイズに加えて 16 KB のブロック・サイズが有効になっていて、6 つのアプリケーション・サーバがあり、それぞれに 2 GB の 8 KB データベース・キャッシュおよび 4 GB の 16 KB データベース・キャッシュがある場合、データ・サーバの 8 KB データベース・キャッシュを調整して制御構造用に 52 MB (50MB + [12 GB * .01]) を使用できるようにし、16 KB キャッシュを調整して制御構造用に 2 MB (24 GB * .005) を使用できるようにする必要があります (どちらの場合も小数点以下は切り上げます)。

データベース・キャッシュの割り当てに関する詳細は、"メモリと開始設定" を参照してください。

2.4.3.3 負荷分散のユーザ要求の影響の評価

ラウンドロビンまたは負荷分散方式でユーザをアプリケーション・サーバに接続すると、アプリケーション・サーバ上でキャッシュから得られる利点が減る可能性があります。ユーザが機能グループに属しており、機能グループで同一のデータを読み取る必要がある場合は、特にこれに該当します。これらのユーザは、複数のアプリケーション・サーバに分散しているため、各アプリケーション・サーバが、データ・サーバに同一のデータを要求する可能性があり、それによって、同じデータに対して複数のキャッシュを使用して分散キャッシュの効率を低下させるだけでなく、ブロックの無効化が増加する可能性もあります。ブロックは、1 つのアプリケーション・サーバで変更されると、他のアプリケーション・サーバでも更新されるためです。これはいくらか主観的ですが、アプリケーションの特性に詳しい人がこの可能性について検討する必要があります。ロード・バランサを構成する場合は、"負荷分散、フェイルオーバー、ミラー構成" を参照して、複数のアプリケーション・サーバにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明を確認してください。

2.4.3.4 トランザクションを単一のデータ・サーバに制限する

単一トランザクション内の更新を、単一のリモート・データ・サーバまたはローカル・サーバに制限します。トランザクションに複数のサーバ (ローカル・サーバを含む) に対する更新が含まれており、TCommit が正常に完了できないと、トランザクションに含まれるサーバによっては、更新をコミットしたものと、ロールバックしたものがある場合があります。詳細は、"コミットの保証" を参照してください。

注釈　IRISTEMP に対する更新は、ロールバックを目的とするトランザクションの一部とは見なされないため、この制限には含まれません。

2.4.3.5 アプリケーション・サーバでの一時グローバルの配置

一時 (スクラッチ) グローバルは、グローバルに共有する必要があるデータを含まない場合、アプリケーション・サーバのローカルに配置する必要があります。多くの場合、一時グローバルは、とてもアクティブで、書き込みが集中します。一時グローバルがデータ・サーバ上に配置されていると、これによって ECP 接続を共有する他のアプリケーション・サーバに不利となります。

2.4.3.6 未定義のグローバルに対する繰り返し参照の回避

未定義のグローバル（例えば、^x が未定義の **$Data(^x(1))** など）が繰り返し参照される場合、グローバルがデータ・サーバで定義されているかどうかをテストするネットワーク処理が常に起こります。

それとは対照的に、定義されたグローバル内の未定義のノード（例えば、^x の他のノードが定義済みの **$Data(^x(1))** など）が繰り返し参照される場合、グローバル（^x）がアプリケーション・サーバ・キャッシュに格納されると、ネットワーク処理は要求されません。

この動作は、ネットワークに接続されていないアプリケーションの場合と大きく異なります。ローカル・データでは、未定義のグローバルに対する参照の繰り返しは、最適化され、不要な作業は排除されます。アプリケーションをネットワーク環境に移植する設計者の場合、グローバルの使用が定義されていたり、定義されていなかったりするため、確認が必要になる場合があります。通常は、グローバルの他の特定のノードが常に定義されていることを確認するだけで十分です。

2.4.3.7 ストリーム・フィールドの使用の回避

ストリーム・フィールドがクエリにあると読み込みロックが発生し、データ・サーバへの接続が必要になります。このことから、このようなクエリではデータベース・キャッシュの利点が得られず、2 回目以降の実行でもパフォーマンスが向上しません。

2.4.3.8 アプリケーション・カウンタでの $Increment 関数の使用

オンライン・トランザクション処理システムの共通の動作は、レコード番号などに使用する一意の値を生成することです。これは、一般的なリレーショナル・アプリケーションで、“次に使用可能な”カウンタ値を含むテーブルを定義して行います。アプリケーションに新規の識別子が必要な場合、カウンタを含む列をロックし、カウンタ値をインクリメントし、ロックを解放します。これは、シングル・サーバ・システムであっても並行処理障害を引き起こします。つまり、アプリケーション・プロセスは、共通カウンタ上のロックが解放されるのを、より長時間待機するようになります。ネットワーク環境では、これはある時点でさらに障害となります。

InterSystems IRIS は、アプリケーション・レベルのロックを一切必要とせずに、（グローバルに格納される）カウンタ値を自動的にインクリメントする $Increment 関数を提供することでこれに対処します。$Increment の並行処理は、InterSystems IRIS データベース・エンジンと ECP の両方に組み込まれており、シングル・サーバや分散アプリケーションで使用した場合に非常に高い効率性が実現されます。

InterSystems IRIS オブジェクト（あるいは SQL）から提供される既定構造を使用して構築されたアプリケーションは、$Increment を自動的に使用して、オブジェクト識別子の値を割り当てます。$Increment は、ECP 上で実行された場合のジャーナル同期に関わる同期処理です。このため、ECP 上での $Increment は比較的低速の処理となり、とりわけ他の比較対象が（アプリケーション・サーバ・データベース・キャッシュまたはデータ・サーバ・データベース・キャッシュのいずれかに）キャッシュ済みのデータを持っているかどうかにより違いが出ます。この影響は、フェイルオーバー・メンバ間のネットワーク遅延により、ミラーリング環境においてさらに大きくなります。このため、アプリケーションを再設計して $Increment を $Sequence 関数に置き換えると便利な場合があります。こうすると、新しい値のバッチは各アプリケーション・サーバ上で各プロセスに自動的に割り当てられ、データ・サーバは新規に値のバッチが必要になった場合にのみ処理に参加します(ただし、連続したアプリケーション・カウンタ値が必要な場合、この手法は使用できません。)$Sequence は $Increment と組み合わせて使用することもできます。

2.4.4 ECP 関連エラー

ECP を使用するシステムには、いくつかの実行時エラーがあります。ECP 関連のエラーは、コマンドの実行直後に発生する場合があります。あるいは、Kill のように、事実上非同期のコマンドの場合、エラーはコマンドの完了後すぐに発生します。

2.4.4.1 <NETWORK> エラー

<NETWORK> エラーは、通常の ECP リカバリ・メカニズムで処理できなかったエラーを示します。

アプリケーション内で <NETWORK> エラーを受け取った場合は常に、プロセスの停止や未実行処理のロールバックが可能です。<NETWORK> エラーの中には、本質的に致命的なエラー状態になるものもあります。すぐに解決する一時的な状態を示すものもあります。しかし、理想的なプログラミング手法とは、<NETWORK> エラーの発生時、未実行処理をロールバックし、現在のトランザクションを最初から開始することです。

$Data や $Order など get 型要求の <NETWORK> エラーは、即座にトランザクションをロールバックするのではなく、手動で再試行できます。ECP は、データを損失する <NETWORK> エラーの発生を避けようとしますが、読み取り専用の要求に多くのエラーを生じます。

2.4.4.2 Rollback Only 条件

アプリケーション側の rollback-only 条件は、アプリケーション・サーバによって開始されたトランザクション中にデータ・サーバがネットワーク障害を検出したときに発生し、トランザクションがロールバックされるまですべてのネットワーク要求でエラーが発生した状態になります。

2.5 ECP リカバリ・プロセス、保証、および制限

ECP リカバリ・プロトコルは、"ECP リカバリ・プロトコル" に要約されています。ここでは、保証と制限を含めて、ECP リカバリについて詳しく説明します。

ECPリカバリの簡単な例は、一時的なネットワーク接続の中断が発生した場合です。この場合の中断時間は、その発生を検知できるほどには長いが、基本の TCP 接続を維持するうえでは問題ない程度に短いと考えます。この障害が発生している間、アプリケーション・サーバは、接続が応答しないことを検知するため、その接続に対する新しいネットワーク要求をブロックします。接続が再開すると、ブロックされていた処理は中断していた要求を送信できるようになります。

基本の TCP 接続がリセットされた場合、データ・サーバは、**[トラブル状態の時間間隔]** 設定 (既定は 1 分) の時間だけ再接続を待ちます。アプリケーション・サーバがこの間隔の間に再接続を成功しないと、データ・サーバは接続をリセットし、開いているトランザクションをロールバックし、ロックを解放します。そのアプリケーション・サーバからの後続の接続は新規の接続要求に変換され、アプリケーション・サーバはその接続がリセットされたことを通知されます。

アプリケーション・サーバは、接続が再確立されると、削除するロックのキューとロールバックするトランザクションを保持します。このキューを保持することにより、中断中のトランザクションやロックがあるデータ・サーバが現在利用可能かどうかにかかわらず、アプリケーション・サーバのプロセスはいつでも停止できます。ECPリカバリでは、データ・サーバのキューに入れられた中断中の Set 処理と Kill 処理をすべて完了してから、ネットワーク障害が検出され、ロックの解放が完了します。

アプリケーション・サーバが接続をリセット (アプリケーション・サーバの再起動などが理由) したことをデータ・サーバが認識するたびに、データ・サーバは、まだ **[トラブル状態の時間間隔]** 内でも即時に接続をリセットし、トランザクションをロールバックし、アプリケーション・サーバの代わりにロックを解放します。アプリケーション・サーバの状態がリセットされるため、代わりにデータ・サーバによって維持される状態はなくなります。

最後の例は、データ・サーバが無理なく、またはクラッシュの結果として停止した場合です。アプリケーション・サーバは、アプリケーションの状態を維持し、**[リカバリまでの待機時間]** 設定 (既定は 20 分) の時間だけデータ・サーバに再接続しようとします。データ・サーバは、クラッシュやシャットダウン時点でアクティブであったアプリケーション・サーバ接続を記憶しています。再起動後、それらのアプリケーション・サーバが再接続され、接続を回復するのを最長で 30 秒待機します。リカバリには、データ・サーバでいくつかの手順が必要です。この手順の中には、非常に重要な方法としてデータ・サーバのジャーナル・ファイルを必要とするものもあります。いくつかの異なる手順の結果は以下のようになります。

- 各アプリケーション・サーバから見た、現在有効なトランザクションのデータ・サーバのビューは、データ・サーバのジャーナル・ファイルからリストアされています。
- 各アプリケーション・サーバから見た、現在有効な Lock オペレーションのデータ・サーバのビューは、アプリケーション・サーバがそれらのロックをデータ・サーバにアップロードすることによってリストアされています。

- アプリケーション・サーバとデータ・サーバは、アプリケーション・サーバからのどの要求を無視し（クラッシュ前に完了したことが確実であるため）、どの要求を再生するかを正確に一致させます。リカバリの最終手順は、単純に中断中のネットワーク要求を完了させることです。ただし、再生しても安全なネットワーク要求のみに限られます。
- 最後に、アプリケーション・サーバはデータ・サーバに、データ・サーバの再起動中に停止したジョブから保存したすべての中断中のアンロックまたはロールバック指示を送ります。ストレージ・デバイス（データベースの場合、WIJ ファイルとジャーナル・ファイル）の整合性が保持されている限り、突然の予期しないデータ・サーバ・クラッシュに直面しても、すべての保証は維持されます。

ECP 構成システムのリカバリ中、InterSystems IRIS は、"ECP リカバリ保証" で詳細に説明されている多数のリカバリ・メカニズムを保証します。これらの保証に対する制限の詳細は、前述の付録の "ECP リカバリの制限" セクションで説明されています。

2.5.1 ECP リカバリ保証

ECP 構成システムのリカバリ中、InterSystems IRIS は以下のリカバリを保証します。

- 更新順の保証
- ECP ロックの保証
- クラスタ・ロックの保証
- ロールバックの保証
- コミットの保証
- トランザクションとロックの保証
- ECP Rollback Only の保証
- ECP トランザクション・リカバリの保証
- ECP ロック・リカバリの保証
- $Increment 順序の保証
- ECP Sync メソッドの保証

各保証についての説明では、最初に具体的な状況を説明します。その後に、その状況に適用できるデータ保証について説明します。

これらの説明では、Process A、Process B などは、データ・サーバでグローバルを更新しようとするプロセスを参照します。これらのプロセスは、同じアプリケーション・サーバまたは異なるアプリケーション・サーバ、あるいはデータ・サーバ自体で生成される可能性があります。プロセスの起源が指定される場合もあれば、関係しない場合もあります。

2.5.1.1 更新順の保証

Process A が 2 つのデータ要素を続けて更新したとします。最初にグローバル ^x を、次にグローバル ^y を更新します。このとき、^x と ^y は、同じデータ・サーバにあります。

Process B は ^y への変更を確認できる場合、^x への変更も確認できます。これは、Process A と Process B が同じアプリケーション・サーバ上にあるかどうかにかかわらず、2 つのデータ・アイテムが同じデータ・サーバにあり、データ・サーバが起動している限り保証されます。

Process B が Process A によって変更されたデータを表示できるからといって、Process A からの Set オペレーションの後に、Process B からの Set オペレーションがリストアされることが保証されるわけではありません。クラスタのフェイルオーバー時またはクラスタのリカバリ時に 2 つの異なるプロセスからの Set や Kill オペレーションが競合する場合、適切な順序を保証できるのは、Lock または $Increment オペレーションのみです。

クラスタのデジャーナル時とクラスタのフェイルオーバー時に異なるプロセスからの Set と Kill オペレーションが競合している場合に適用される順序については、"クラスタのフェイルオーバー時やリストア時の緩やかな順序" の制限を参照してください。

重要　この保証は、データ・サーバがクラッシュした場合、^x と ^y がジャーナルされている場合でも、適用されません。この説明に該当するプロセスで、データ・サーバがクラッシュする以前に保存されなかったダーティ・データが読み取れるケースについては、"ECP でのロックしないダーティ・データの読み取り" 制限を参照してください。

2.5.1.2 ECP ロックの保証

Process B (DataServer S 上) が、Process A によってロックされたことがあるグローバル ^x のロックを取得したとします。

Process B は、(^x へのロックを保持している間) Process A によって行われたすべての更新を DataServer S で表示できますまた、Process C が、(^x へのロックを保持している間) Process B によって行われた更新をDataServer S で表示した場合、Process C は、(^x へのロックを保持している間) Process A によって行われた更新も DataServer S で表示することが保証されます。

このシリアル化機能は、Process A、Process B、Process C が同じアプリケーション・サーバ、または DataServer S 自体に属するかどうかにかかわらず、DataServer S がその間常に起動している限り保証されます。

重要　保護されるロックとデータは、同じデータ・サーバ上にある必要があります。

2.5.1.3 クラスタ・ロックの保証

クラスタ・メンバの Process B が、Process A によってロックされたことがあるクラスタ・データベースのグローバル ^x へのロックを取得したとします。

この場合、Process B は、(^x へのロックを取得している間、) Process A によっていずれかのクラスタ・データベースに行われたすべての更新を表示できます。

また、クラスタ・メンバの Process C が、(^x へのロックを取得した場合、) Process B によってクラスタ・データベースに行われた更新を表示できるだけでなく、Process C は、(^x へのロックを保持した状態で、) Process A によっていずれかのクラスタ・データベースに行われた更新も表示できます。

このシリアル化機能は、Process A、Process B、Process C が同じクラスタ・メンバに属すかどうか、またはクラスタ・メンバがクラッシュしているかどうかにかかわらず保証されます。

重要　あるクラスタ・メンバのトランザクションが、クラッシュしたクラスタ・メンバのトランザクションから、ダーティ・データを表示するケースについては、"クラスタ・メンバのクラッシュ時のダーティ・データの読み取り" の制限を参照してください。

2.5.1.4 ロールバックの保証

Process A が、一連の更新の後に、TStart コマンドを実行してから、TCommit を発行する前に中断したか、TCommit を実行する前に TRollback を実行したとします。

Process A により、トランザクションの一部として行われた更新はすべて、最初とは逆の順序でロールバックされます。

重要　ロールバックの制限の詳細は、"非ロック状態の変更により競合が発生した場合のロールバック"、"ジャーナルが中断した場合のロールバック"、および "非同期のエラーが TCommit で検出された場合のロールバック" を参照してください。

2.5.1.5 コミットの保証

Process A が DataServer S で一連の更新を行い、TCommit の実行を開始した後に中断したとします。

トランザクションに含まれる DataServer S それぞれで、DataServer S でのデータの変更がコミットまたはロールバックされます。TCommit を実行するプロセスの Perform Synchronous Commit プロパティがオンになっている場合に (構成ファイル内で SynchCommit=1 に設定)、TCommit オペレーションから正常に値が返されると、そのトランザクションは、トランザクションの一部であるすべてのデータ・サーバで確実にコミットされたことが保証されます。

重要　トランザクションに複数のサーバ (ローカル・サーバを含む) に対する更新が含まれており、TCommit が正常に完了できないと、トランザクションに含まれるサーバによっては、更新をコミットしたものと、ロールバックしたものがある場合があります。

2.5.1.6 トランザクションとロックの保証

Process A では Transaction T のために TStart を実行し、DataServer S でグローバル ^x をロックした後、^x をロック解除します (ロック解除しても、"即時アンロック" ロック・タイプは指定されません)。

InterSystems IRIS では、^x へのロックは、トランザクションがコミットされるかロールバックされるまで解除されないことが保証されます。Transaction T が DataServer S でコミットするか、ロールバックするまで、他のプロセスは、^x へのロックを取得できません。

Transaction T が DataServer S でコミットすると、Process B では、^x へのロックを取得し、Transaction T の間に、Process A によって DataServer S に行われた変更が表示されます。また、他のプロセスが、(^x へのロックを保持している間) Process B によって DataServer S に行われた変更を表示した場合、(Transaction T 実行中に) Process A によって DataServer S に行われた変更が表示されます。逆に、Transaction T が DataServer S でロールバックした場合、^x へのロックを取得した Process B が、Process A によって DataServer S に行われた変更を表示しようとしても、何も表示されません。

重要　詳細は、"非ロック状態の変更により競合が発生した場合のロールバック" を参照してください。

2.5.1.7 ECP Rollback Only の保証

Process A (AppServer C 上) が、Transaction T の一部である変更を DataServer S で行い、DataServer S が一方的にその変更をロールバックします (これは、特定のネットワーク障害やデータ・サーバ障害で発生する場合がある)。

その後 Process A による DataServer S に対するネットワーク要求はすべて、Process A が明示的に TRollback コマンドを実行しない限り、<NETWORK> エラーによって拒否されます。

また、AppServer C のプロセスが、DataServer S のロールバックと Transaction T の TCommit の間に、DataServer S へのネットワーク要求を完了した場合 (AppServer C が TCommit の前に rollback-only 条件を検出した場合)、Transaction T には、Transaction T の一部であるすべてのデータ・サーバにロールバックが行われることが保証されます

2.5.1.8 ECP トランザクション・リカバリの保証

データ・サーバが、アプリケーション・サーバのトランザクションの途中でクラッシュし、その後再起動して、アプリケーション・サーバのリカバリ・タイムアウト時間内にリカバリを完了します。

この場合、トランザクションは、保証されているとおりに、正常に完了できます。データ・サーバは、ロック定義によって定義されている順番の制約に違反するようなデータ処理は実行しません。唯一の例外は、$Increment 関数です (詳細は、"ECP とクラスタの $Increment 制限" を参照してください)。リカバリできないトランザクションは、ロック定義を保護するようにロールバックされます。

重要　(ネットワーク、データ・サーバ、アプリケーション・サーバのハードウェアやソフトウェアなどで) 障害が継続して発生していない場合、InterSystems IRIS では、データ・サーバ停止時にデータ・サーバで未実行のすべてのあるいは大半のトランザクションは復元できるはずです。しかし、これは保証されません。

2.5.1.9 ECP ロック・リカバリの保証

DataServer S に計画外のシャットダウンが発生したため、再起動して、リカバリ時間内にリカバリを完了しました。

この場合、ECP ロックの保証は、変更されたデータがジャーナルされている限り適用されます。データがジャーナルされていない場合、クラッシュの前のデータ・サーバに対する更新は、アプリケーション・サーバに通知されることなく、失われます。InterSystems IRIS では、ロックを取得したプロセスが、そのロックを保持している間に、他のプロセスによって以前に行われたすべての更新を表示できることは保証されません。

DataServer S を正しくシャットダウンし、再起動し、リカバリ時間内にリカバリを完了した場合は、データがジャーナルされているかどうかにかかわらず、ECP ロックの保証は適用されます。

トランザクションの一部である更新は、常にジャーナルされ、ECP トランザクション・リカバリの保証は確実に適用されます。他のジャーナルは、目的のデータベース内のグローバルがジャーナリングとマークされているかどうかによって、ジャーナルされる場合とされない場合があります。

2.5.1.10 $Increment 順序の保証

$Increment 関数には、異なるプロセスからの Set や Kill オペレーションがロックで保護されていなくても、それらの一連のオペレーションに対する緩やかな順序があります。

例えば、Process A が、DataServer S で、Set と Kill オペレーションを実行し、DataServer S でグローバル ^x に $Increment オペレーションを実行します。Process B は、その後に、同じグローバル ^x に $Increment を実行します。Process B を含むどのプロセスからでも、^x をインクリメントした Process B の結果を表示する場合、^x をインクリメントする前に Process A が DataServer S に実行したすべての変更も表示されます。

重要　　詳細は、"ECP とクラスタの $Increment 制限" のセクションを参照してください。

2.5.1.11 ECP Sync メソッドの保証

プロセス A は、データ・サーバ S にあるグローバルを更新し、$system.ECP.Sync() 呼び出しを S に対して発行します。続いて、プロセス B は $system.ECP.Sync() を S に対して発行します。プロセス B は、$system.ECP.Sync() 呼び出しの前に、プロセス A がデータ・サーバ S で実行したすべての更新を表示できます。

$system.ECP.Sync() は、アプリケーション・サーバで実行しているプロセスにのみ関係があります。プロセス A または B のいずれかがデータ・サーバ S 自体で実行している場合、そのプロセスは $system.ECP.Sync() を発行する必要はありません。両方のプロセスがデータ・サーバ S で実行している場合、どちらも $system.ECP.Sync を必要としません。これは、同じサーバで実行しているプロセスがグローバル更新を直ちに見ることができることを示す文に過ぎません。

重要　　$system.ECP.Sync() は持続性を保証しません。"ECP でのロックしないダーティ・データの読み取り" の制限を参照してください。

2.5.2 ECP リカバリの制限

ECP 構成システムのリカバリ中、InterSystems IRIS の保証には以下の制限があります。

- ECP とクラスタの $Increment 制限
- ECP Cache の最新性の制限
- ECP ルーチンの再検証の制限
- 非ロック状態の変更により競合が発生した場合のロールバック
- ジャーナルが中断した場合のロールバック
- ECP のリカバリ後に検出されないエラー
- 部分的 Set または Kill によるジャーナルの不一致

- クラスタのフェイルオーバーまたはリストア時の緩やかな順序
- クラスタ・メンバのクラッシュ時のダーティ・データの読み取り
- ECP でのロックしないダーティ・データの読み取り
- 非同期のエラーが TCommit で検出された場合のロールバック

2.5.2.1 ECP とクラスタの $Increment 制限

データ・サーバに対する未実行 $Increment 要求がアプリケーション・サーバにあり、グローバルがジャーナリングされたとき、データ・サーバがクラッシュすると、InterSystems IRIS は、$Increment 結果をジャーナルから回復しようとし、参照を再びインクリメントしません。

2.5.2.2 ECP Cache の最新性の制限

障害が継続して発生していない場合は、アプリケーション・サーバは数秒以内にデータの更新を参照できますが、これは保証されていません。特に、ECP のデータ・サーバとの接続が中断した場合 (ネットワーク障害、データ・サーバのシャットダウン、データ・サーバのバックアップ処理など)、ユーザ・プロセスはアプリケーション・サーバの接続タイムアウト値を限度に古いデータを参照する場合があります。データが古くないことを確認するには、Lock コマンドをデータ・フェッチ操作で使用するか、または $system.ECP.Sync を使用します。アプリケーション・サーバとデータ・サーバ間を往復するネットワーク要求によって、アプリケーション・サーバ ECP ネットワーク・キャッシュのコンテンツが更新されます。

2.5.2.3 ECP ルーチンの再検証の制限

アプリケーション・サーバがデータ・サーバからルーチンをダウンロードして、データ・サーバが (計画的または計画外に) 再起動されると、データ・サーバからダウンロードされたルーチンは編集済みであるかのようにマークされます。

また、データ・サーバとの接続にネットワーク障害 (アプリケーション・サーバかデータ・サーバのシャットダウン) が発生した場合も、データ・サーバからダウンロードされたルーチンは編集済みであるかのようにマークされます。この動作は、場合によっては、実体を伴わない <EDITED> エラーや <ERRTRAP> エラーが発生させることがあります。

2.5.2.4 非ロック状態の変更により競合が発生した場合のロールバック

InterSystems IRIS では、Lock コマンドはアドバイスに過ぎません。Process A が、グローバル ^y へのロックが保護された状態で、グローバル ^x を更新するトランザクションを開始し、別のプロセスが、^y へのロックが保護されていない状態で、^x を変更した場合、^x のロールバックは機能しません。

Set と Kill オペレーションのロールバックでは、データ・アイテムの現在の値がオペレーションによって設定された値である場合、その値は、オペレーションの前の値にリセットされます。現在の値が Set や Kill オペレーションによって設定された値とは異なる場合、現在の値は変更されません。

データ・アイテムが、ある時点でトランザクションの内部で変更され、別の時点でトランザクションの外部で、Lock コマンドで保護されずに変更されるような場合は、ロールバック機能は保証されません。ロールバックを有効にするには、データ・アイテムを変更する場合に必ずロックを使用する必要があります。

2.5.2.5 ジャーナルが中断した場合のロールバック

ロールバックは、ジャーナルの信頼性と完全性に依存しています。ジャーナルのデータが何かによって中断された場合、その中断した時点以降のロールバックは行えません。InterSystems IRIS では、このようなトランザクションのロールバックは暗黙的に無視されます。

ジャーナルの中断が発生する可能性がある場合は、InterSystems IRIS の動作中に ^JRNSTOP を実行した場合、InterSystems IRIS がシャットダウンした後や再起動する前に Write Image Journal (WIJ) ファイルを削除した場合、またはジャーナルのエラー時にシステムをフリーズするように設定されていないシステムで、ジャーナリング中に入出力エラーが発生した場合です。

2.5.2.6 ECP のリカバリ後に検出されないエラー

Set や Kill オペレーションが、データ・サーバで完了した後、エラーが発生したとします。データ・サーバはそのパケットの処理は完了していますが、アプリケーション・サーバ・システムに送信する前にクラッシュします。

この場合、ECP リカバリではそのパケットを再生しませんが、アプリケーション・サーバはそのエラーを検出していないため、アプリケーション・サーバは、データ・サーバ側の Set または Kill オペレーションを失います。

2.5.2.7 部分的 Set または Kill によるジャーナルの不一致

Set や Kill オペレーションを問題なくジャーナルできても、実際にデータベースを変更する前にエラーが発生するケースがあります。データのロールバックを定義する特別な方法があっても、それによって、トランザクションのロールバックが行われなくなることはありませんが、ジャーナルのリストア後のデータベースの状態は、リストア前のデータベースの状態と一致しなくなる場合があります。

2.5.2.8 クラスタのフェイルオーバーまたはリストア時の緩やかな順序

クラスタのデジャーナリングは緩やかに順序付けられます。異なるクラスタ・メンバのジャーナル・ファイルが同期化されるのは、ロック、$Increment、またはジャーナル・マーカー・イベントが発生した場合のみです。これは、クラスタ全体を停止してリストアする必要があるクラスタのフェイルオーバーやクラスタのクラッシュ後のデータベース状態に影響します。その場合、データベースは、クラッシュ前の状態と異なる状態にリストアされる場合があります。$Increment 順序の保証には、リストアされたデータベースがクラッシュ前の元の状態と異なる度合いに関する新たな制約があります。

Process B が Process A によって変更されたデータを表示できるからといって、Process A からの Set オペレーションの後に、Process B からの Set オペレーションがリストアされることが保証されるわけではありません。クラスタのフェイルオーバー時またはクラスタのリカバリ時に 2 つの異なるプロセスからの Set や Kill オペレーションが競合する場合、適切な順序を保証できるのは、Lock または $Increment オペレーションのみです。

2.5.2.9 クラスタ・メンバのクラッシュ時のダーティ・データの読み取り

クラスタ・メンバ Member A が Transaction T1 で更新を完了し、そのトランザクションが非同期のトランザクション・コミット・モードでコミットされたとします。別のクラスタ Member B の Transaction T2 が、Transaction T1 によって所有されていたロックを取得したとします。そして、Transaction T1 からのすべての情報がディスクに書き込まれる前に、クラスタ・メンバ Member A がクラッシュしたとします。

この場合、Transaction T1 は、クラスタ・フェイルオーバーの一部としてロールバックされます。しかし、Member B の Transaction T2 では、ロック・プロトコルの規則に従って、Transaction T1 の情報を表示できたはずであるのに、後でクラスタ・フェイルオーバーの一部としてロールバックされています。また、Transaction T2 で Transaction T1 と同じデータ・アイテムが一部変更された場合、Transaction T1 のロールバックは、一部のトランザクション・データしかロールバックできないため、失敗する可能性があります。

これを回避する方法は、クラスタ・メンバ Member A のトランザクションを同期コミット・モードで行うことです。同期コミット・モードを使用した場合、Transaction T1 はロックが解除される前にディスクに保存され、アプリケーションが Transaction T1 が完全であることを認識すれば、ロールバックされることはありません。

2.5.2.10 ECP でのロックしないダーティ・データの読み取り

ECP トランザクションがロックしないでデータを読み取る場合、ディスクに保存されていないデータを読み取りますが、データ・サーバがクラッシュするとそのデータは失われる可能性があります。そのようなデータを読み取れるのは、他の ECP 接続やローカル・データ・サーバ・システム自体によって、データの場所が設定されている場合のみです。保存されていないデータが、その接続自体によって設定されている場合は表示できません。また、データを読み取るプロセスとデータを書き込むプロセスの両方にロックが使用されている場合、保存されていないデータを表示することはできません。ここでは、更新順の保証が適用されず、ロックを使用する以外に簡単に解決できる方法はありません。

2.5.2.11 非同期のエラーが TCommit で検出された場合のロールバック

データ・サーバ側のトランザクションが、データベースの更新中に、<FILEFULL> などの非同期のエラー状況に遭遇し、アプリケーション・サーバが TCommit 時までそのエラーを認識できなかった場合、トランザクションは自動的にデータ・サーバ側でロールバックされます。ただし、TCommit オペレーションは通常は非同期ですが、ロールバックは同期です。これは、アプリケーション・サーバ・プロセスがロックを解除するまでにアプリケーション・サーバに通知しておく必要があるブロックが、ロールバックによって変更されるためです。

データ・サーバとデータベースには問題はありませんが、アプリケーション・サーバ側では、ロックが別のプロセスに移った場合、アプリケーション・サーバはロールバックされようとするデータを一時的に参照する場合があります。しかし、アプリケーション・サーバでは、非同期エラーを発生させることを通常は行いません。

3

シャーディングによるデータ量に応じた水平方向の拡張

この章では、InterSystems IRIS シャード・クラスタの導入と使用について説明します。

3.1 InterSystems IRIS のシャーディングの概要

シャーディングは、InterSystems IRIS データ・プラットフォームを水平方向に拡張するための重要な機能です。InterSystems IRIS シャード・クラスタは、データの格納とキャッシュの両方を複数のサーバ間で分割して、クエリおよびデータ取り込みに応じた柔軟で安価なパフォーマンス拡張を実現しながら、きわめて効率的にリソースを使用することにより、インフラストラクチャの価値を最大化します。シャーディングは、InterSystems IRIS の優れた垂直方向の拡張性と容易に組み合わせることができ、InterSystems IRIS がソリューションを提供する作業負荷の範囲が大幅に広がります。

- シャーディングの要素
- シャーディングの効果の評価
- ネームスペース・レベルのシャーディング・アーキテクチャ

注釈　シャード・クラスタを導入および使用するための実践演習を含むシャーディングの簡単な紹介については、"InterSystems IRIS デモ：シャード・クラスタの導入" を参照してください。

3.1.1 シャーディングの要素

シャーディングによる InterSystems IRIS の水平方向の拡張は、さまざまなアプリケーションに有益ですが、以下のいずれかまたは両方を含むユース・ケースで最も大きな効果が得られます。

- ディスクからの大量のデータの取得、データの複雑な処理、またはその両方 (分析作業負荷など)
- 大量で高速のデータ取り込み

シャーディングは、データ・ノードと呼ばれる複数の InterSystems IRIS インスタンス間で大規模なデータベース・テーブルおよびそれらに関連付けられたインデックスを水平方向に分割すると同時に、これらのインスタンスのいずれかを介してアプリケーションがこれらのテーブルにアクセスできるようにします。データ・ノードが集まって、シャード・クラスタを形成します。このアーキテクチャには以下の利点があります。

- シャード・テーブルに対するクエリは、すべてのデータ・ノード上で並列で実行され、結果はマージおよび集約され、完全なクエリ結果としてアプリケーションに返されます。

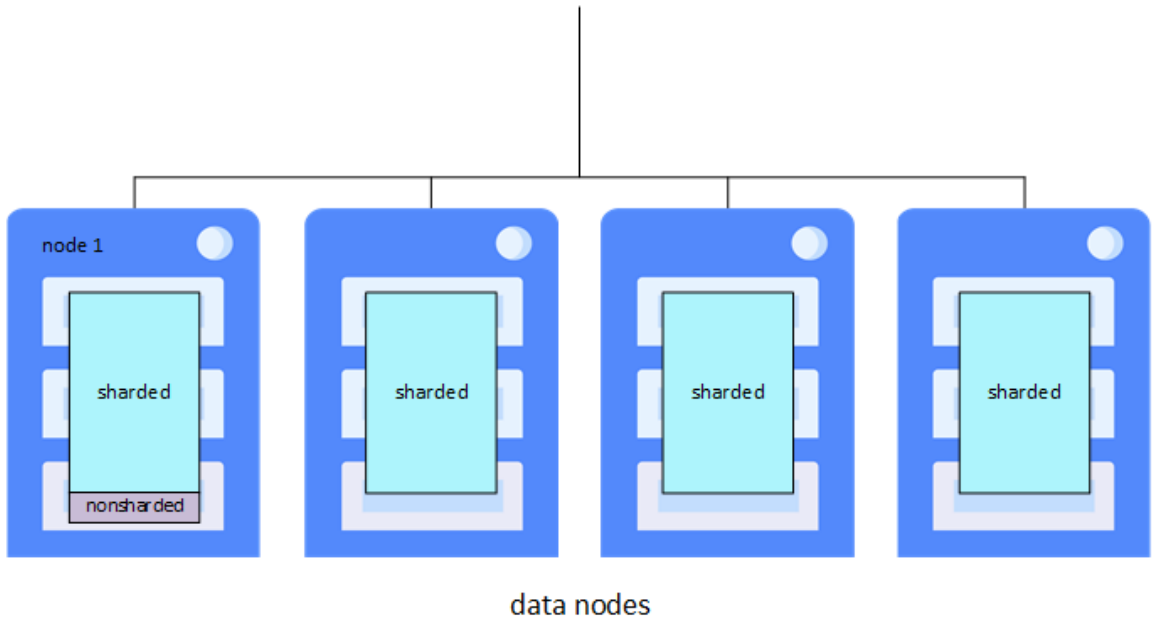
- データ分割は別個のインスタンスによってホストされるため、単一インスタンスのキャッシュがデータ・セット全体を処理するのではなく、データ・セットの独自のパーティションを処理する専用キャッシュがあります。

シャーディングにより、大きなテーブルに対するクエリのパフォーマンスが、単一システムのリソースによって制約されなくなります。複数のシステムにわたってクエリ処理とキャッシュの両方を分散することで、シャーディングは計算リソースとメモリ・リソースの両方の線形に近い拡大を実現し、作業負荷に合わせて調整されたクラスタを設計および管理できます。データ・ノードを追加することによりスケール・アウトすると、シャード・データをクラスタ全体で再分散することができます。この分散データ・レイアウトは、データの並列ロードやサードパーティ・フレームワークでの使用などで活用することもできます。

シャードは、テーブルの行のサブセットです。各行は 1 つのシャード内に格納され、テーブルのすべてのシャードにほぼ同じ数の行が格納されます。各データ・ノードは、クラスタ上のシャード・テーブルごとに 1 つのシャードで構成されるデータ・シャードをホストします。シャーディング・マネージャと呼ばれるフェデレートされたソフトウェア・コンポーネントは、どのシャード (およびどのテーブル行) がどのデータ・ノードにあるかを追跡し、それに応じてクエリを送ると共に、他のシャード操作を管理します。各テーブルは、そのフィールドの 1 つをシャード・キーとして使用することで、自動的にデータ・ノード間で水平方向に分割されます。シャード・キーは、データを均等に分散するための確実な方法を提供します。シャード・キーは一般にはテーブルの RowId (既定) ですが、ユーザ定義のフィールドまたはフィールド・セットにすることもできます。

シャード・データはデータ・ノード間で物理的に分割されますが、任意のデータ・ノードで (シャード化されていないデータ、メタデータ、およびコードとして) すべて論理的に表示可能です。各データ・ノードには、クラスタ上のすべてのデータおよびコードに透過的にアクセスできるようにする、クラスタ・ネームスペース (クラスタ全体で同一の名前) があります。アプリケーションは任意のノードのクラスタ・ネームスペースにアクセスでき、全データセットをローカルのものであるかのように利用できます。したがってアプリケーション接続はクラスタ内すべてのデータ・ノードで負荷分散され、クエリの並列処理や分割されたキャッシュを最大限に活用する必要があります。

図 3-1: 基本的なシャード・クラスタ



シャード化されていないデータは、最初に構成されたデータ・ノード (データ・ノード 1、または単にノード 1 と呼ぶ) にのみ格納されます。この区別は、より多くのデータがノード 1 に格納されるということ以外、ユーザに対して透過的です。ただし、この違いも一般的には小さいものです。アプリケーション SQL から見ると、シャード・テーブルとシャード化されていないテーブルの区別は透過的です。

InterSystems IRIS ミラーリングを使用すると、シャード・クラスタ内のデータ・ノードに対して高可用性を実現できます。単一インスタンスと同じように簡単に InterSystems IRIS インスタンスのミラー・フェイルオーバー・ペアをクラスタに追加できます。ミラーリングされたシャード・クラスタの導入の詳細は、"ミラー・データ・ノードによる高可用性" を参照してください。

常に大量のデータが取り込まれるような状況でも、クエリの遅延がきわめて小さいことが求められる高度な使用事例では、計算ノードを追加してクエリを処理するための透過的なキャッシュ層を提供することで、クエリとデータ取り込みの作業負荷を分離し、両方のパフォーマンスを向上させることができます。1 つのデータ・ノードに複数の計算ノードを割り当てることで、クラスタのクエリ・スループットをさらに向上させることができます。計算ノードとそれらの導入手順の詳細は、"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" を参照してください。

3.1.2 シャーディングの効果の評価

InterSystems IRIS のシャーディングは、さまざまなアプリケーションに有益ですが、以下を含むユース・ケースで最も大きな効果が得られます。

- 比較的大きなデータ・セット、大量のデータを返すクエリ (またはその両方)。

  シャーディングでは、データと共にキャッシュを分割することにより、複数のシステムのメモリ・リソースを利用し、データのサイズに合わせてキャッシュの処理能力を調整します。単一インスタンスのデータベース・キャッシュがすべてのデータに使用可能であるのと比べて、各データ・ノードのデータベース・キャッシュ (グローバル・バッファ・プール) はデータ・セットの一部のみに使用されます。定期的に使用されるデータが大きすぎて、シャード化されていない単一インスタンスのデータベース・キャッシュに収まらない場合、パフォーマンスの向上が最も顕著になります。

- 大量のデータ処理を行う大量の複雑なクエリ。

  シャーディングでは、クエリを分解し、複数のデータ・ノード間で並列で実行することにより、複数のシステムの計算リソースを利用し、クエリ処理のスループットを調整します。クラスタに対するクエリが以下の条件に該当する場合、パフォーマンスの向上が最も顕著です。

  - 永続ストレージから大量のデータを読み取り、特に、返す結果に対して取得するデータの割合が高い。
  - かなりの計算作業 (集約、グループ化、ソートなど) を必要とする。

- 大容量または高速のデータ取り込み (またはその組み合わせ)。

  シャーディングでは、InterSystems IRIS JDBC ドライバを使用してデータ・ノードに直接接続し、並列でデータをロードすることにより、複数のインスタンス間で取り込みを分散し、データ取り込みを調整します。データが検証済みであるとして、一意性チェックを省略できる場合は、効果がさらに大きくなります。

これらの要因はそれぞれ個別でも、シャーディングから得られる潜在的恩恵に影響しますが、結合すれば恩恵も大きくなる場合があります。例えば、大量のデータの高速の取り込み、大規模データ・セット、大量のデータを取得して処理する複雑なクエリを組み合わせると、現代の分析上の作業負荷の多くがシャーディングに非常に適した候補になります。

前述のように、これまでに説明したいくつかの要因に対処するために InterSystems IRIS のシャーディングを垂直方向の拡張と組み合わせて使用すれば、さまざまな状況で最も高い効果が得られる可能性があります (詳細は、"InterSystems IRIS シャード・クラスタの計画" を参照してください)。

注釈　現在のリリースでは、シャーディングで、アトミック性を必要とする複雑なトランザクションが関与する作業負荷をサポートしておらず、そのような作業負荷に対してシャード・クラスタを使用することはできません。

3.1.3 ネームスペース・レベルのシャーディング・アーキテクチャ

このドキュメントの以前のバージョンでは、より大規模な別のノード・タイプのセット (シャード・マスタ・データ・サーバ、シャード・データ・サーバ、シャード・クエリ・サーバ) を含むシャーディング・アーキテクチャについて説明していました。このネームスペース・レベルのアーキテクチャは、新しいノード・レベルのアーキテクチャの透過的な基盤として残っており、ノード・レベルのアーキテクチャと完全な互換性があります。ノード・レベルのアーキテクチャにおいて、クラスタ・ネームスペース (クラスタ全体で同一の名前) は、クラスタ上のすべてのデータおよびコード (シャード化されているもの、いな

いもの共に) への透過的なアクセスを提供します。最初のデータ・ノードに配置されるようになったマスタ・ネームスペースは、依然としてメタデータ、シャード化されていないデータ、およびコードへのアクセスを提供し、すべてのデータ・ノードで完全に利用可能です。これにより、導入および使用が単純で便利な、より統一され、簡単なモデルが提供されます。

管理ポータルの [シャード構成] ページおよび %SYSTEM.Sharding を使用して、ネームスペース・レベルのシャード・クラスタを導入できます。これらの手順は、"ネームスペース・レベル・アーキテクチャの導入" で説明されています。

3.2 シャード・クラスタの導入

評価やテストのために最初のクラスタを作成する簡単な方法を提供するため、このセクションでは、ミラーリングされていないデータ・ノードで構成される基本的な InterSystems IRIS シャード・クラスタを導入し、管理ポータルまたは %SYSTEM.Cluster を使用してそのクラスタを構成する手動手順を説明します。それ以降の各セクションでは、以下のように、これらの手順を拡張して計算ノードとミラーリングを追加します。

- 計算ノードは既存のクラスタに簡単に追加できます。プロダクション環境への計算ノードの導入を検討する場合、一般的には、データ・ノードのみのクラスタの動作を評価してから、クラスタにとって計算ノードの追加が有益であるかどうかを決定することをお勧めします。計算ノードの計画と追加の詳細は、"計算ノードの計画" および "作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" を参照してください。
- 各データ・ノードにフェイルオーバー機能 (およびオプションの災害復旧機能) を追加すること、およびフェイルオーバー・メンバ間の遅延によって軽微な影響が出る可能性があることを除くと、ミラーリングされたシャード・クラスタの動作は、同じ数のデータ・ノードのミラーリングされていないクラスタとまったく同じです。ミラーリングされたシャード・クラスタの導入に関心がある場合、手順については "ミラー・データ・ノードによる高可用性" を参照してください。

InterSystems IRIS データ・プラットフォームでは、シャード・クラスタの自動導入方法もいくつか提供されています。これらの方法を使うと、オンプレミス・ハードウェア、パブリック・クラウド、プライベート・クラウド、Kubernetes などのさまざまなトポロジのクラスタを導入するプロセスが大幅に簡素化されます。

シャード・クラスタの導入にどの方法を使用するかにかかわらず、最初のステップでは、クラスタに含めるデータ・ノードの数を決定し、それらのノード上のデータベース・キャッシュとグローバル・データベースのサイズを計画します。手動で導入する場合は、クラスタを構成する前に、クラスタをホストするインフラストラクチャを特定またはプロビジョニングし、ホストに InterSystems IRIS インスタンスを導入する必要もあります。したがって、このセクションの手動手順を使用して基本のクラスタを導入するには、以下のように、最初にこのセクションのステップを実行してから、手動構成方法を選択します。

1. データ・ノードの計画
2. データベース・キャッシュとデータベースのサイズの見積もり
3. インフラストラクチャのプロビジョニングまたは特定
4. データ・ノード・ホストへの InterSystems IRIS の導入
5. 以下のいずれかを使用したクラスタの構成
   - 管理ポータル
   - %SYSTEM.Cluster API

重要　このセクションの手順では、以下に注意してください。

- ミラーリングされたシャード・クラスタまたは計算ノードの導入については説明されていません。ミラーリングされた導入については "ミラーによる高可用性" で説明しています。一方、"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" では、API を使用して基本のクラスタに計算ノードを追加する方法を説明しています。
- ノード・レベルのシャード・クラスタを導入します。ネームスペース・レベルのクラスタの導入手順は、"ネームスペース・レベル・アーキテクチャの導入" を参照してください。

注釈　これらの手順では、クラスタを構成する前にプロビジョニングまたは特定したホストに新しい InterSystems IRIS インスタンスを導入することを前提としていますが、既存のインスタンスでも使用できるように手順を変更できます。ただしその場合、クラスタ・ノードとして構成するインスタンスとそのホストが、最初の 4 つのステップで説明する要件とガイドラインに準拠していることが条件です。

複数のデータ・ノードおよび (該当する場合は) 計算ノードにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明は、"Web ゲートウェイ・ガイド" の "負荷分散、フェイルオーバー、ミラー構成" を参照してください。

メモリ管理および拡張、CPU サイジングおよび拡張、その他の考慮事項を含む、パフォーマンス計画の重要な説明は、"システム・リソースの計画と管理" を参照してください。

最も標準的なシャード・クラスタ構成では、各クラスタ・ノードは 1 つの物理または仮想システム上の 1 つの InterSystems IRIS インスタンスで構成されます。この章で説明する手順では、この構成が前提となります。

LDAP サーバを使用して、シャード・クラスタのノード全体で一元的なセキュリティを実装することをお勧めします。InterSystems IRIS での LDAP の使用についての詳細は、"LADP ガイド" を参照してください。

HealthShare Health Connect ではシャーディングはサポートされません。

3.2.1 クラスタの自動導入方法

このセクションで概要を説明する手動手順のほかにも、InterSystems IRIS データ・プラットフォームでは、導入後に完全に機能するシャード・クラスタの自動導入方法が 2 つ提供されています。

3.2.1.1 InterSystems Kubernetes Operator (IKO) を使用したシャード・クラスタの導入

Kubernetes は、コンテナ化されたワークロードとサービスの導入、拡張、および管理を自動化するためのオープンソースのオーケストレーション・エンジンです。導入するコンテナ化されたサービスと、そのサービスを管理するポリシーを定義すると、Kubernetes は、必要なリソースを可能な限り最も効率的な方法で透過的に提供します。また、導入が指定値から外れた場合は導入を修復またはリストアするほか、拡張を自動またはオンデマンドで行います。InterSystems Kubernetes Operator (IKO) は、IrisCluster カスタム・リソースで Kubernetes API を拡張します。このリソースは、InterSystems IRIS のシャード・クラスタ、分散キャッシュ・クラスタ、またはスタンドアロン・インスタンスとして、すべて任意でミラーリングした状態で、Kubernetes プラットフォームに導入できます。

Kubernetes で InterSystems IRIS を導入するのに IKO は必須ではありませんが、プロセスが大幅に簡易化され、InterSystems IRIS 固有のクラスタ管理機能が Kubernetes に追加され、クラスタにノードを追加するなどのタスクが可能になります。このようなタスクは、IKO を使わなければインスタンスを直接操作して手動で行わなければなりません。

IKO の使用法の詳細は、"InterSystems Kubernetes Operator の使用" を参照してください。

3.2.1.2 構成マージを使用したシャード・クラスタの導入

Linux および UNIX® システムで利用可能な構成マージ機能を使用すると、宣言型構成マージ・ファイルを導入内の各インスタンスに適用することにより、同じイメージから導入した InterSystems IRIS コンテナや、同じキットからインストールしたローカル・インスタンスの構成を変更することができます。

このマージ・ファイルは、既存のインスタンスを再起動したときに適用することもでき、インスタンスの構成パラメータ・ファイル（CPF）を更新します。CPF にはインスタンスのほとんどの構成設定が含まれており、これらの設定は開始時に毎回、インスタンス導入後の最初の設定を含めて CPF から読み取られます。導入時に構成マージを適用すると、インスタンスと共に提供された既定の CPF が実質的に独自の更新バージョンに置き換えられます。

構成マージを使用すると、ノード・タイプごとに別個のマージ・ファイルを呼び出してシャード・クラスタを導入できます。その際、データ ノード 1、次に残りのデータ・ノード、そして計算ノード（オプション）というように順番に導入できます(他のデータ・ノードを構成するには、データ・ノード 1 として構成されているインスタンスが実行されている必要があるため、他のインスタンスを導入する前に、このインスタンスを導入して正常に開始する必要があります)。導入ホストの名前が指定した正規表現で終わっていれば、1 つのマージ・ファイルを使用して、データ・ノードのクラスタを自動的に導入（およびオプションでミラーリング）することもできます。データ・ノードの導入後、オプションで別のマージ・ファイルを使用して計算ノードを導入できます。

前述のように、IKO には構成マージ機能が組み込まれています。

一般的な構成マージの使用法および具体的なシャード・クラスタの導入方法の詳細は、"構成マージを使用した InterSystems IRIS の自動構成" を参照してください。

3.2.2 データ・ノードの計画

クラスタに格納するシャード・データについて予測される作業セットと、そのデータに対して実行するクエリの性質に応じて、クラスタに適したデータ・ノードの数がわずか 4 台の場合もあります。いつでもデータ・ノードを既存のクラスタに追加し、シャード・データ を再分散できるため（"データ・ノードの追加とデータの再分散" を参照)、控えめになるのはもっともです。

(リソースの制限を条件として) プロダクション構成に必要なデータ・ノードの最適な数を最初に見積もるのに適した基本的な方法は、クラスタに必要なデータベース・キャッシュの総量を計算した後、状況およびリソースの可用性に応じて、サーバの数とサーバごとのメモリ量をどのように組み合わせるのが構成に適しているかを判断することです。これは、複数のシステム間で必要なリソースを分割する必要がある場合を除き、通常のサイジング・プロセスと同様です(メモリ管理および拡張、CPU サイジングおよび拡張、その他の考慮事項を含む、パフォーマンス計画の重要な説明は、"システム・リソースの計画と管理" を参照してください)。

必要なデータベース・キャッシュのサイズを決定するには、クラスタ上に格納することが予想されるシャード・データの総量、および頻繁にシャード・データと結合されるクラスタ上のシャード化されていないデータの量を見積もることから開始します。次にこれらの合計を使用して、シャード・データと頻繁に結合されるシャード化されていないデータの両方の作業セットを見積もり、これらを合わせたものが、クラスタ内のすべてのデータ・ノードで必要なデータベースのキャッシュ処理能力の合計を表します。この計算の詳細は、"InterSystems IRIS シャード・クラスタの計画" を参照してください。

ノード数とノードごとのメモリの両方に関するすべてのオプションを検討したら、各データ・ノード上のデータベース・キャッシュ（グローバル・バッファ・プール）がその処理能力の割り当て分に等しくなる、またはほぼ等しくなるように、十分なデータ・ノードを構成できます。さまざまなシナリオで、開始時のデータ・ノード数は、必要なキャッシュ・サイズの合計を、クラスタ・ノードとして導入するために使用できるシステムのメモリ容量で割ることで、大まかに決定することができます。

シャード・クラスタ内のすべてのデータ・ノードの仕様とリソースが同じであるか、少なくともほぼ同等である必要があります。クエリの並列処理の速度は、最も低速なデータ・ノードと同じ速度にしかなりません。さらに、クラスタ内のすべての InterSystems IRIS インスタンスの構成が一貫している必要があります。確実に正しい SQL クエリ結果が返されるようにするには、インスタンス・レベルで構成される照合などのデータベース設定やそれらの SQL 設定（既定の日付形式など）がすべてのノードで同じでなければなりません。標準化された手順や、シャード・クラスタで利用可能な自動導入方法を使用すると、この一貫性を簡単に確保できます。

アプリケーションは任意のデータ・ノードのクラスタ・ネームスペースに接続して、全データセットをローカルのものであるかのように利用できるので、一般的なベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード間でアプリケーション接続を負荷分散することをお勧めします。IKO は、一般的なシナリオの場合、必要に応じてデータ・ノードのロード・バランサを自動的にプロビジョニングし、構成します。他の手段でシャード・クラスタを導入する場合は負荷分散メカニズムが必要となります。複数のデータ・ノードにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明は、"Web ゲートウェイ・ガイド" の "負荷分散、フェイルオーバー、ミラー構成" を参照してください。

3.2.3 データベース・キャッシュとデータベースのサイズの見積もり

シャード・クラスタを導入する前に、各データ・ノードに割り当てるデータベース・キャッシュのサイズを決定します。これは、予想される増大に十分対応できる空き容量を確保できるよう、各データ・ノードの既定のグローバル・データベースに必要なデータ量の予想サイズを把握するのにも有効です。

自動導入方法を使用してシャード・クラスタを導入する場合、これらの設定を導入の一部として指定できます。シャーディング API または管理ポータルを使用して手動で導入する場合は、シャード・クラスタを構成する前に各インスタンスのデータベース・キャッシュ・サイズを指定し、呼び出しでデータベース設定を指定することができます。両方の手動導入方法で既定の設定を提供しています。

以下に示すサイズはガイドラインであり、要件ではないこと、また、実践ではこれらの数値の見積もりが調整される可能性があることに留意してください。

3.2.3.1 データベース・キャッシュ・サイズ

"InterSystems IRIS シャード・クラスタの計画" で説明しているように、理想的にデータ・ノード上のデータベース・キャッシュに割り当てる必要があるメモリ量は、予想されるシャード・データの作業セットすべてのノードへの割り当て分に、頻繁にシャード・データに結合されるシャード化されていないデータの予想される作業セットすべてを加えたものになります。

3.2.3.2 グローバル・データベース・サイズ

"InterSystems IRIS シャード・クラスタの計画" で説明しているように、既定のグローバル・データベースのターゲット・サイズは以下のとおりです。

- クラスタ・ネームスペース ー シャード・データの合計サイズのうちの各サーバへの割り当て分 (上述のセクションで説明している計算による) に、予想を超える増大を見越したマージンを加えたもの。
- ノード 1 のマスタ・ネームスペース ー シャード化されていないデータの合計サイズに、予想を超える増大を見越したマージンを加えたもの。

すべての導入方法でこれらのデータベースのサイズが既定で構成されているため、ユーザがこの構成を行う必要はありません。ただし、これらのデータベースが配置されるストレージが、ターゲット・サイズに対応できる大きさであることを確認する必要があります。

3.2.4 インフラストラクチャのプロビジョニングまたは特定

必要なネットワーク・ホスト・システム (物理、仮想、またはクラウド) の数をプロビジョニングまたは特定します (各データ・ノードにつき 1 つのホスト)。

重要

シャード・クラスタ内のすべてのデータ・ノードの仕様とリソースが同じであるか、少なくともほぼ同等である必要があります。クエリの並列処理の速度は、最も低速なデータ・ノードと同じ速度にしかなりません (計算ノードでも同じことが言えます。ただし、その場合、ストレージは考慮されません)。

ベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード間でアプリケーション接続を負荷分散することをお勧めします。

クラスタのパフォーマンスを最大限に発揮させるためのベスト・プラクティスは、すべてのデータ・ノード間に低遅延のネットワーク接続を構成してネットワーク・スループットを最大化することです。例えば、すべてのデータ・ノードを同じデータ・センターまたはアベイラビリティ・ゾーン内の同じサブネットに配置します。この手順では、データ・ノードが TCP/IP 経由で相互にアクセス可能であることが前提となっており、推奨ネットワーク帯域幅はすべてのノード間で最小 1 GB、利用可能であれば 10 GB 以上が推奨されています。

3.2.5 データ・ノード・ホストへの InterSystems IRIS の導入

この手順では、各システムで単一の InterSystems IRIS インスタンスをホストするか、またはホストする予定であることが前提となっています。各インスタンスとそのホストがこのステップで説明する要件を満たす限り、最後のステップで新しく導入したインスタンスの代わりに既存のインスタンスを構成できます。

重要　シャード・クラスタ内のすべての InterSystems IRIS インスタンスは、同じバージョンである必要があり、シャーディング・ライセンスを有する必要があります。

可能であれば、すべてのインスタンスについて、データベース・ディレクトリとジャーナル・ディレクトリを別個のストレージ・デバイスに配置してください。これは、大量のデータ取り込みがクエリの実行と同時に行われる場合に特に重要です。ファイル・システムの構成およびジャーナル・ストレージなどのストレージの構成のガイドラインについては、"システム・リソースの計画と管理" の "ストレージの計画" と "ファイル・システムの分離"、および "データ整合性ガイド" の "ジャーナリングの最善の使用方法" を参照してください。

クラスタ内のすべての InterSystems IRIS インスタンスの構成が一貫している必要があります。確実に正しい SQL クエリ結果が返されるようにするには、インスタンス・レベルで構成される照合などのデータベース設定やそれらの SQL 設定 (既定の日付形式など) がすべてのノードで同じでなければなりません。

各ホスト・システムで、以下を実行します。

1. インターシステムズが提供するイメージからコンテナを作成する ("コンテナ内でのインターシステムズ製品の実行" を参照) か、キットから InterSystems IRIS をインストールする ("インストール・ガイド" を参照) ことにより、InterSystems IRIS のインスタンスを導入します。
2. "データベース・キャッシュとデータベースのサイズの見積もり" の説明に従って、インスタンスのデータベースをホストするストレージ・デバイスがグローバル・データベースのターゲット・サイズに十分対応できる大きさであることを確認します。
3. "データベース・キャッシュとデータベースのサイズの見積もり" で決定したサイズに従って、インスタンスのデータベース・キャッシュ (グローバル・バッファ・プール) を割り当てます。管理ポータルでデータベース・キャッシュを割り当てる手順は、"システム管理ガイド" の "“InterSystems IRIS の構成”" の章にある "メモリと開始設定" を参照してください。また、インスタンスの構成パラメータ・ファイル (CPF) を編集するか、UNIX® および Linux プラットフォームでは構成マージ・ファイルを使用して目的の値でインスタンスを導入することにより、globals パラメータを使用してキャッシュを割り当てることもできます。

   クラスタ・メンバの共有メモリ・ヒープのサイズを大きくすることがお勧めの手段となることもあります。共有メモリ・ヒープは、"構成パラメータ・ファイル・リファレンス" の gmheap パラメータの説明のように管理ポータルを使用して編集するか、上記の globals の説明のように gmheap を使用してインスタンスの CPF ファイルで編集できます。

   注釈　InterSystems IRIS インスタンスのルーチン・キャッシュとデータベース・キャッシュおよび共有メモリ・ヒープに必要なメモリを見積もる際の一般的なガイドラインは、"メモリ要件の見積もり" を参照してください。

最後に、管理ポータル (次のセクション) または %SYSTEM.Cluster API (次の次のセクション) を使用して、導入したインスタンスをシャード・クラスタとして構成します。

3.2.6 管理ポータルを使用したクラスタの構成

最初の 4 つのステップ (データ・ノードの計画、データベース・キャッシュとデータベース・サイズの見積もり、インフラストラクチャのプロビジョニングまたは特定、およびデータ・ノード・ホストへの InterSystems IRIS の導入) を完了したら、この手順を使用して、前のステップで導入したインスタンス (または準備しておいた既存のインスタンス) を、管理ポータルを使用してデータ・ノードの基本の InterSystems IRIS シャード・クラスタとして構成します。

以下のステップを使用して、クラスタ内の各ノードを構成します。

1. ノード 1 の構成
2. 残りのデータ・ノードの構成

ブラウザで管理ポータルを開く方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎：IDE の接続" の "InterSystems IRIS 接続情報" のセクションで、キットからインストールしたインスタンスの手順を参照してください。

この手順の後に、他の管理ポータルのシャード・クラスタ・オプションについて簡単に説明します。

3.2.6.1 データ・ノード 1 の構成

シャード・クラスタは最初のデータ・ノードを構成するときに初期化されます。最初のデータ・ノードをデータ・ノード 1、または単にノード 1 と呼びます。このデータ・ノードは、クラスタのシャード化されていないデータ、メタデータ、およびコードを格納し、すべてのデータ・ノードがそのデータにアクセスできるようにするマスタ・ネームスペースをホストするという点で、他のノードとは異なります。この区別は、より多くのデータがこの最初のノードに格納されるということ以外、ユーザに対して完全に透過的です。ただし、この違いも一般的には小さいものです。

最初の 4 つのステップ (データ・ノードの計画、データベース・キャッシュとデータベース・サイズの見積もり、インフラストラクチャのプロビジョニングまたは特定、およびデータ・ノード・ホストへの InterSystems IRIS の導入) を完了したら、この手順を使用して、前のステップで導入したインスタンス (または準備しておいた既存のインスタンス) を、管理ポータルを使用してデータ・ノードの基本の InterSystems IRIS シャード・クラスタとして構成します。

ノード 1 を構成するには、以下の手順を実行します。

1. インスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→** [Sharding]**→[シャード有効]** の順に選択し、表示されるダイアログで [OK] をクリックします(既定値はほぼすべてのクラスタに適しているため、**[ECP 接続の最大数]** 設定の値を変更する必要はありません)。
2. インスタンスを再起動します(管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動した後に再読み込みするだけでかまいません)。
3. [Configure Node-Level] ページ (**[システム管理]→[構成]→[システム構成]→**[Sharding]**→**[Configure Node-Level]) に移動して、**[構成]** ボタンをクリックします。
4. [Configure Node-Level Cluster] ダイアログで [Initialize a new sharded cluster on this instance] を選択し、表示されるプロンプトに次のように応答します。
   - [Cluster namespace] および **[マスターネームスペース]** のドロップダウンからそれぞれクラスタ・ネームスペースとマスタ・ネームスペースを選択します。どちらのドロップダウンにも以下が含まれます。
     - 作成する新しいネームスペースの既定の名前 (**IRISCLUSTER** および **IRISDM**) と、その既定のグローバル・データベース。
     - 対象となるすべての既存のネームスペース。

     シャード・クラスタを初期化すると、既定で、クラスタとマスタ・ネームスペース (それぞれ **IRISCLUSTER** および **IRISDM** という名前) と共に、その既定のグローバル・データベースと必要なマッピングが作成されます。ただし、クラスタ・ネームスペースとマスタ・ネームスペースの名前、およびそのグローバル・データベースの特性を制御するため、クラスタを構成する前に一方または両方のネームスペースと既定のデータベースを作成しておき、この手順でそれらを指定できます。例えば、"グローバル・データベース・サイズ" で説明されている考慮事項を踏まえ、クラスタ・ネームスペースの既定のグローバル・データベースの特性や、クラスタ内のすべてのデータ・ノードに複製されるシャード・データベースを制御するために、この方法を実行できます。

     注釈　クラスタ・ネームスペースとして指定した既存ネームスペースの既定グローバル・データベースになんらかのグローバルまたはルーチンがあると、初期化がエラーで失敗します。

- 場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、**ホスト名のオーバーライド**のプロンプトで IP アドレスを入力します。
- [Enable mirroring] は選択しないでください。ミラーリングされたシャード・クラスタの導入手順は、"ミラーによる高可用性" で説明されています。

5. [OK] をクリックして [Configure Node-Level] ページに戻ります。2 つのタブ **[シャード]** と **[シャードテーブル]** が表示されています。ノード 1 がクラスタ・アドレス (次の手順で必要になります) を含めて **[シャード]** に表示されるので、参照用にノード 1 の管理ポータルで [Configure Node-Level] ページを開いたままにしておくことができます。**[シャードを検証]** をクリックして、ノード 1 が正しく構成されていることを検証します。

3.2.6.2 残りのデータ・ノードの構成

ノードを構成したら、以下の手順を使用して追加の各データ・ノードを構成します。

1. インスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→** [Sharding]**→[シャード有効]** の順に選択し、表示されるダイアログで [OK] をクリックします(既定値はほぼすべてのクラスタに適しているため、**[ECP 接続の最大数]** 設定の値を変更する必要はありません)。
2. インスタンスを再起動します(管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動した後に再読み込みするだけでかまいません)。
3. [Configure Node-Level] ページ (**[システム管理]→[構成]→[システム構成]→**[Sharding]**→**[Configure Node-Level]) に移動して、**[構成]** ボタンをクリックします。
4. [Configure Node-Level Cluster] ダイアログで [Add this instance to an existing sharded cluster] を選択し、表示されるプロンプトに次のように応答します。
   - クラスタ URL を入力します。これは、[Configure Node-Level] ページの **[シャード]** タブにノード 1 に対して表示されているアドレスです (前の手順の説明を参照してください)。クラスタ URL は、ノード 1 ホストの ID (ホスト名、または前の手順で IP アドレスを指定した場合は IP アドレス) に、InterSystems IRIS インスタンスのスーパーサーバ・ポートとクラスタ・ネームスペースの名前を加えたものになります。例えば、clusternode011.acmeinternal.com:1972:IRISCLUSTER のようになります(クラスタ・ネームスペースの名前は、既定の **IRISCLUSTER** である場合は省略できます)。

     注釈　もう 1 つのノード (この手順で必要なノード) から見ると、コンテナ化された InterSystems IRIS インスタンスのスーパーサーバ・ポートは、そのコンテナが作成されたときにスーパーサーバ・ポートが公開されたホスト・ポートによって異なります。この詳細および例は、"コンテナ内でのインターシステムズ製品の実行" の "永続的な %SYS を使用した InterSystems IRIS コンテナの実行" と "InterSystems IRIS コンテナの実行 : Docker Compose の例"、および Docker ドキュメントの "Container networking" を参照してください。

     キットでインストールした、そのホスト上にしかない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。複数のインスタンスがインストールされている場合、スーパーサーバ・ポート番号は 51776 以上の範囲になります。インスタンスのスーパーサーバ・ポート番号を表示または変更するには、管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します。
   - **[ロール]** プロンプトで **[データ]** を選択して、インスタンスをデータ・ノードとして構成します。
   - 場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、**ホスト名のオーバーライド**のプロンプトで IP アドレスを入力します。
   - [Mirrored cluster] は選択しないでください。ミラーリングされたシャード・クラスタの導入手順は、"ミラーによる高可用性" で説明されています。

5. [OK] をクリックして [Configure Node-Level] ページに戻ります。2 つのタブ **[シャード]** と **[シャードテーブル]** が表示されています。これまでに構成したデータ・ノードが **[シャード]** にノード 1 から表示されます。**[シャードを検証]** をクリックして、データ・ノードが正しく構成されていて、他のノードと通信できることを検証します。

   注釈　構成するデータ・ノードが多数ある場合は、**[詳細設定]** ボタンをクリックし、[詳細設定] ダイアログで **[割り当て時に自動的にシャードを検証]** を選択することにより、検証操作を自動化できます。シャード・クラスタを導入する際は、このダイアログのその他の設定は既定値のままにします。

3.2.6.3 管理ポータルのシャーディング・オプション

[Rebalance] ボタンを使用すると、[REBALANCE] ダイアログが表示されます。このダイアログで、既存のクラスタに最近追加されたデータ・ノードを含むすべてのデータ・ノードに均等にシャード・データを再分散できます(再分散の実行中にシャード・テーブルをクエリすることはできません)。データの再分散の詳細は、"データ・ノードの追加とデータの再分散"を参照してください。

[詳細設定] ダイアログは、[Configure Node-Level] ページの **[詳細設定]** ボタンを押すと表示できます。このダイアログには以下のオプションがあります。

- **[割り当て時に自動的にシャードを検証]** を選択すると、新しいクラスタ・ノードの構成後の検証を自動化できます。
- クラスタのシャード・クエリ実行モードを既定の非同期から同期に変更します。この効果の説明については、"ミラーリングされたクラスタに計算ノードを含めることによる、フェイルオーバー前後での透過的なクエリ実行"を参照してください。
- シャードの接続タイムアウトを既定値の 60 秒から変更します。
- ホスト名ではなく、データ・ノード 1 ホストの IP アドレスを使用してクラスタ通信中に特定します。
- クラスタがミラー・シャードへの接続を再試行する回数を既定値の 1 から変更します。
- ローカル・ノードの ECP 接続の最大数を変更します。クラスタ内のノードの合計数がこの設定を超えることはできません。このセクションの 2 つのステップそれぞれの開始時に説明したように、この設定は、管理ポータルの **[シャード有効]** オプションを使用した場合、64 に設定されます。
- クラスタを、DROP TABLE 操作中にエラーを無視するように設定します。

[ノードレベルの構成] ページの **[シャードテーブル]** タブに、クラスタ上のすべてのシャード・テーブルに関する情報が表示されます。

3.2.7 %SYSTEM.Cluster API を使用したクラスタの構成

以下の手順を使用して、前のステップで %SYSTEM.Cluster API を使用してデータ・ノードの基本の InterSystems IRIS シャード・クラスタとして導入したインスタンス(または準備しておいた既存のインスタンス)を構成します。"インターシステムズ・クラス・リファレンス" の %SYSTEM.Cluster クラス・ドキュメントを参照してください(%SYSTEM パッケージのすべてのクラスと同様に、**%SYSTEM.Cluster** メソッドは、**$SYSTEM.Cluster** を介しても使用できます)。

以下のステップを使用して、クラスタ内の各ノードを構成します。

1. ノード 1 の構成
2. 残りのデータ・ノードの構成

3.2.7.1 ノード 1 の構成

シャード・クラスタは最初のデータ・ノードを構成するときに初期化されます。最初のデータ・ノードをデータ・ノード 1、または単にノード 1 と呼びます。このデータ・ノードは、クラスタのシャード化されていないデータ、メタデータ、およびコードを格納し、すべてのデータ・ノードがそのデータにアクセスできるようにするマスタ・ネームスペースをホストするという点

で、他のノードとは異なります。この区別は、より多くのデータがこの最初のノードに格納されるということ以外、ユーザに対して完全に透過的です。ただし、この違いも一般的には小さいものです。

ノード 1 を構成するには、インスタンスの InterSystems ターミナルを開き、$SYSTEM.Cluster.Initialize() メソッドを呼び出します。以下に例を示します。

```
set status = $SYSTEM.Cluster.Initialize()
```

注釈　これらの手順で詳述されている各 API 呼び出しの戻り値 (成功の場合は 1 など) を確認するには、以下を入力します。

```
zw status
```

状況によっては通知なしで呼び出しが失敗することがあるため、各呼び出しの後に **[ステータス]** を確認することをお勧めします。呼び出しが成功しなかった (**[ステータス]** が [1] 以外) 場合、以下を入力することにより、わかりやすいエラー・メッセージが表示されます。

```
do $SYSTEM.Status.DisplayError(status)
```

Initialize() 呼び出しにより、マスタ・ネームスペースとクラスタ・ネームスペース (それぞれ **IRISDM** と **IRISCLUSTER**)、およびその既定のグローバル・データベースを作成し、必要なマッピングを追加します。ノード 1 はクラスタの他の部分に対してテンプレートとして機能します。クラスタ・ネームスペースの名前、既定のグローバル・データベース (シャード・データベースとも呼ばれる) の特性、およびマッピングは、2 番目に構成したデータ・ノードに直接複製され、さらにその他すべてのデータ・ノードに直接、または間接的に複製されます。インスタンスの SQL 構成設定も複製されます。

クラスタとマスタ・ネームスペースの名前、およびそれらのグローバル・データベースの特性を制御するには、クラスタのネームスペース、マスタ・ネームスペース、またはその両方として既存のネームスペースを指定 (いずれかまたは両方の名前を引数として含めることによって指定) します。以下に例を示します。

```
set status = $SYSTEM.Cluster.Initialize("CLUSTER","MASTER",,)
```

この際、指定した各ネームスペースの既存の既定グローバル・データベースは残ります。これにより、シャード・データベースの特性を制御でき、これらはクラスタ内の他のデータ・ノードに複製されます。

注釈　クラスタ・ネームスペースとして指定した既存ネームスペースの既定グローバル・データベースになんらかのグローバルまたはルーチンがあると、初期化がエラーで失敗します。

既定では、どのホストもクラスタ・ノードになることができます。Initialize() の 3 番目の引数で、IP アドレスまたはホスト名のコンマ区切りのリストを指定することで、クラスタに参加できるホストを指定できます。リストにないノードはクラスタに参加できません。

場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、4 番目の引数として IP アドレスを含めます(この引数にホスト名を指定することはできません。IP アドレスのみです)。いずれの場合も、2 番目のデータ・ノードを構成する際、ホスト識別子 (ホスト名または IP アドレス) を使用してノード 1 を識別します。インスタンスのスーパーサーバ (TCP) ポートも必要です。

注釈　もう 1 つのノード (この手順で必要なノード) から見ると、コンテナ化された InterSystems IRIS インスタンスのスーパーサーバ・ポートは、そのコンテナが作成されたときにスーパーサーバ・ポートが公開されたホスト・ポートによって異なります。この詳細および例は、"コンテナ内でのインターシステムズ製品の実行" の "永続的な %SYS を使用した InterSystems IRIS コンテナの実行" と "InterSystems IRIS コンテナの実行 : Docker Compose の例"、および Docker ドキュメントの "Container networking" を参照してください。

キットでインストールした、そのホスト上にしかない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。インスタンスのスーパーサーバ・ポート番号を表示または設定するには、そのインスタンスの管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します (ブラウザで管理ポータルを開く方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎 : IDE の接続" の "InterSystems IRIS 接続情報" のセクションで、キットからインストールしたインスタンスの手順を参照してください)。

Initialize() メソッドは、InterSystems IRIS インスタンスが既にシャード・クラスタのノードであるか、ミラー・メンバである場合、エラーを返します。

3.2.7.2 残りのデータ・ノードの構成

各追加データ・ノードを構成するには、InterSystems IRIS インスタンスのターミナルを開き、既存のクラスタ・ノード (2 番目のノードを構成している場合はノード 1) のホスト名とその InterSystems IRIS インスタンスのスーパーサーバ・ポートを指定して $SYSTEM.Cluster.AttachAsDataNode() メソッドを呼び出します。以下に例を示します。

```
set status = $SYSTEM.Cluster.AttachAsDataNode("IRIS://datanode1:1972")
```

ノード 1 の初期化の際、Initialize() の 4 番目の引数として IP アドレスを指定した場合は、最初の引数でノード 1 を識別するためにホスト名の代わりに IP アドレスを使用します。以下に例を示します。

```
set status = $SYSTEM.Cluster.AttachAsDataNode("IRIS://100.00.0.01:1972")
```

注釈　指定する正しいスーパーサーバ・ポートの特定についての重要な情報は、前の手順、"ノード 1 の構成" を参照してください。

AttachAsDataNode() 呼び出しは、以下を実行します。

- クラスタ・ネームスペースとシャード・データベースを作成し、それらを "ノード 1 の構成" で説明したとおり、テンプレート・ノードの設定 (最初の引数で指定) に合わせて構成し、必要なマッピングを作成して、これらをノード 1 のマスタ・ネームスペースのグローバル・データベースとルーチン・データベースに含めます (ユーザ定義のマッピングを含む)。
- すべての SQL 構成オプションを、テンプレート・ノードに合わせて設定します。
- このノードは AttachAsDataNode() のテンプレート・ノードとして後で使用される可能性があるため、クラスタに参加できるホストのリストを、ノード 1 の Initialize() 呼び出しで指定したリスト (ある場合) に設定します。

注釈　テンプレート・ノードのクラスタ・ネームスペースと同じ名前のネームスペースが新しいデータ・ノードに存在する場合は、そのネームスペースとそのグローバル・データベースがクラスタ・ネームスペースとシャード・データベースとして使用され、マッピングのみが複製されます。続いてこの新しいノードがテンプレート・ノードとして使用される場合は、これらの既存の要素の特性が複製されます。

AttachAsDataNode() 呼び出しは、InterSystems IRIS インスタンスが既にシャード・クラスタのノードまたはミラー・メンバであるか、最初の引数で指定されたテンプレート・ノードがミラー・メンバである場合、エラーを返します。

前のステップで説明したように、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させるには、IP アドレスを 2 番目の引数として含めます(この引数にホスト名を指定することはできません。IP アドレスのみです)。

すべてのデータ・ノードを構成したら、$SYSTEM.Cluster.ListNodes() メソッドを呼び出してこれらをリストできます。以下に例を示します。

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    Host          Port
1       Data        datanode1     1972
2       Data        datanode2     1972
3       Data        datanode3     1972
```

この例のように、データ・ノードには、クラスタにアタッチされる順序を表す数値 ID が割り当てられます。

ベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード間でアプリケーション接続を負荷分散することをお勧めします。

クラスタへの計算ノードの追加の詳細は、"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" を参照してください。

3.3 シャード・テーブルの作成とデータのロード

クラスタが完全に構成されたら、シャード・テーブルを計画および作成し、それらにデータをロードします。必要な手順は以下のとおりです。

- シャーディングに関する既存のテーブルの評価
- シャード・テーブルの作成
- クラスタへのデータのロード
- シャード化されていないテーブルの作成とロード

3.3.1 シャーディングに関する既存のテーブルの評価

クラスタ上のシャード化されていないデータに対するシャード・データの割合は一般には高いですが、既存のスキーマのシャード・クラスタへの移行を計画する際は、すべてのテーブルがシャーディングに適した候補であるとは限らないことに留意してください。アプリケーションのテーブルのうち、どれをシャード・テーブルとして定義するのか、どれをシャード化されていないテーブルとして定義するのかを決定する際には、主に、以下の要因に基づいて、クエリ・パフォーマンスやデータ取り込みの速度の向上を検討する必要があります (これらの要因については、"シャーディングの効果の評価" でも説明しています)。計画の際、シャード・テーブルとシャード化されていないテーブルの区別は、アプリケーション SQL に対して完全に透過的であることに留意してください。インデックスの選択と同様に、シャーディングの決定はパフォーマンスにのみ影響します。

- 全体のサイズ – 他の条件がすべて同じであれば、テーブルが大きいほど、効果が大きくなる可能性があります。
- データ取り込み – 頻繁または大規模な (あるいはその両方の) INSERT 文をテーブルが受け取るかどうか。データの並列ロードは、シャーディングによってそのパフォーマンスを向上させることができることを意味します。
- クエリの量 – 継続的に最も頻繁にクエリが実行されているテーブルはどれか。この場合も、他の条件がすべて同じであれば、クエリの量が多いほど、パフォーマンスが大きく向上する可能性があります。
- クエリ・タイプ – クエリの量が多い、大きいテーブルの中でも、多くのデータを読み取る (特に、返す結果に対して読み取るデータの割合が高い) クエリを頻繁に受け取るものや多くの計算作業を行うものは、シャーディングに適した候補となります。例えば、幅広い SELECT 文によって頻繁にスキャンされるテーブルや、集約関数を含む多数のクエリを受け取るテーブルなどです。

シャーディングに適した候補をいくつか特定したら、以下の考慮事項について確認します。

- 頻繁な結合 – "シャード・キーの選択" で説明するように、頻繁に結合されるテーブルは、同じシャード・キーでシャード化してコシャード結合を有効にすることができます。これにより、個々のシャード上でローカルに結合を実行できるため、パフォーマンスが向上します。等値条件の 2 つの大きいテーブルを結合する、頻繁に使用される各クエリを調べて、コシャード結合の機会を示しているかどうかを評価します。テーブルのコシャーディングによって効果が得られるクエリが全体的なクエリ作業負荷のかなりの部分を占めている場合、これらの結合されるテーブルはシャーディングに適した候補となります。

  ただし、大きいテーブルがはるかに小さいものに頻繁に結合されるときに、大きいテーブルをシャード化し、小さいテーブルをシャード化されないようにすると、最も効果的な場合があります。シャード化するテーブルを選択する際には、特定の結合の頻度やクエリ・コンテキストを入念に分析すると非常に役立ちます。

- 一意制約 – シャード・キーが一意キーのサブセットでない限り、シャード・テーブルの一意制約によって、挿入/更新のパフォーマンスに重大な悪影響が生じることがあります。詳細は、"シャード・キーの選択" を参照してください。

重要　　他の要因に関係なく、アトミック性を必要とする複雑なトランザクションに含まれるテーブルはシャード化しないでください。

3.3.2 シャード・テーブルの作成

シャード・テーブル (およびシャード化されていないテーブル) は、任意のノードのクラスタ・ネームスペースで、シャーディング仕様を含む SQL CREATE TABLE 文を使用して作成できます。このシャーディング仕様は、テーブルがシャーディングされること、および使用されるシャード・キー (シャード・テーブルのどの行をどのシャードに格納するかを特定するフィールド) を示します。シャード間でテーブルの行を均等に分散するための確実な方法を提供する適切なシャード・キーでテーブルが作成されると、INSERT や専用ツールを使用してここにデータをロードできます。

3.3.2.1 シャード・キーの選択

既定では、シャード・テーブルを作成してシャード・キーを指定していない場合、システムにより割り当てられた RowID をシャード・キーとして使用して、データがここにロードされます。例えば、2 つのシャードがある場合、RowID=1 の行は一方のシャードに、RowID=2 の行はもう一方のシャードにロードされます。これはシステムが割り当てたシャード・キー (SASK) と呼ばれ、データの均等分散を最適に保証し、最も効率的な並列データ・ロードを可能にするため、多くの場合、最も簡単で効果的なアプローチになります。

注釈　　既定では、RowID フィールドは **ID** と名付けられ、列 1 に割り当てられます。**ID** というユーザ定義フィールドが追加されると、テーブルのコンパイル時に RowID フィールドは **ID1** という名前に変更され、キーを指定せずにシャーディングを行う際、既定で使用されるのは、このユーザ定義の **ID** フィールドになります。

シャード・テーブルの作成時に、シャード・キーとして 1 つ以上のフィールドを指定するオプションもあります。これはユーザ定義のシャード・キー (UDSK) と呼ばれます。スキーマに RowID に対応しない意味的に重要な一意の識別子が含まれている場合、例えば、スキーマ内のいくつかのテーブルに **accountnumber** フィールドがある場合は、UDSK を使用する良い機会かもしれません。

その他の考慮事項として、大きなテーブルを結合するクエリがあります。各シャード・クエリはシャードローカル・クエリに分解されます。それぞれのクエリは個々に、各シャード上でローカルに実行され、そのシャードに存在するデータを参照すればよいだけです。ただし、シャード・クエリに 1 つ以上の結合が含まれる場合、シャードローカル・クエリは一般に他のシャードのデータを参照する必要があるため、処理時間が長くなると共に、データベース・キャッシュに割り当てられたメモリをより多く使用します。この余分なオーバーヘッドは、コシャード結合を有効にすることで回避できます。コシャード結合では、結合される 2 つのテーブルの行が同じシャード上に配置されます。結合がコシャードされている場合、その結合が関与するクエリは、同じシャード上の行のみを結合するシャードローカル・クエリに分解され、他のシャード・クエリと同様に個別にローカルに実行されます。

コシャード結合は、以下の 2 つのアプローチのいずれかを使用して有効にできます。

- 2 つのテーブルに対して同等な UDSK を指定する。

- 1 つのテーブルに SASK を使用し、もう 1 つのテーブルで coshard with キーワードと適切な UDSK を使用する。

同等な UDSK を使用するには、2 つのテーブルにシャード・キーとして頻繁に結合されるフィールドを指定するだけです。例えば CITATION テーブルと VEHICLE テーブルを結合し、各車両に関連付けられた交通違反通知を返す場合、以下のようにします。

```
SELECT * FROM citation, vehicle where citation.vehiclenumber = vehicle.vin
```

この結合をコシャード結合にするには、同等なそれぞれのフィールドをシャード・キーとして使用して、両方のテーブルを作成します。

```
CREATE TABLE VEHICLE (make VARCHAR(30) not null, model VARCHAR(20) not null,
  year INT not null, vin VARCHAR(17) not null, shard key (vin))

CREATE TABLE CITATION(citationid VARCHAR(8) not null, date DATE not null,
  licensenumber VARCHAR(12) not null, plate VARCHAR(10) not null,
  vehiclenumber VARCHAR(17) not null, shard key (vehiclenumber))
```

シャーディング・アルゴリズムは決定的であるため、特定の VIN の VEHICLE 行と CITATION 行 (ある場合) (それぞれ vin および vehiclenumber フィールドの値) は、同じシャード上に配置されます (ただし、フィールド値そのものが、行の各セットがどのシャードに配置されるかを特定することはありません)。したがって、上記のクエリが実行されると、各シャードローカル・クエリは、ローカルに (つまり、完全にそのシャード上で) 結合を実行できます。シャード・キーとして使用される 2 つのフィールド間の等値条件が含まれていなければ、結合をこのようにコシャードすることはできません。同様に、それぞれのテーブルのシャード・キーに、フィールド値の等値性を比較できるタイプの同じフィールド番号が同じ順序で存在する限り、複数フィールドの UDSK を使用して、コシャード結合を有効にできます。

多くのケースで有効なその他のアプローチでは、SASK を使用して 1 つのテーブルを作成し、最初のテーブルと共にコシャードされることを示す coshard with キーワード、および最初のテーブルのシステムが割り当てた RowID 値に等しい値を持つシャード・キーを指定することにより、もう 1 つのテーブルを作成します。例えば、以下のように、クエリで ORDER テーブルおよび CUSTOMER テーブルを頻繁に結合しているとします。

```
SELECT * FROM orders, customers where orders.customer = customers.%ID
```

この場合、結合の片方のフィールドは RowId を表すため、テーブル CUSTOMER を SASK を使用して次のように作成することから開始します。

```
CREATE TABLE CUSTOMER (firstname VARCHAR(50) not null, lastname VARCHAR(75) not null,
  address VARCHAR(50) not null, city VARCHAR(25) not null, zip INT, shard)
```

コシャード結合を有効にするには、CUSTOMER テーブルへの参照として customer フィールドが定義されている ORDER テーブルを、そのフィールドで CUSTOMER テーブルとのコシャードを指定することで、シャーディングします。

```
CREATE TABLE ORDER (date DATE not null, amount DECIMAL(10,2) not null,
  customer CUSTOMER not null, shard key (customer) coshard with CUSTOMER)
```

前に説明した UDSK の例と同様、これにより、ORDER の各行は、RowID 値がその customerid 値と一致する CUSTOMER の行と同じシャード上に配置されます (例えば、customerid=427 であるすべての ORDER 行は、ID=427 の CUSTOMER 行と同じシャード上に配置されます)。このようにして有効になったコシャード結合は、SASK でシャーディングされたテーブルの ID と、そのテーブルと共にコシャードされるテーブルに対して指定されるシャード・キーとの間の等値条件を含む必要があります。

一般には、スキーマで指定した以下のいずれかを使用することで、最も有用なコシャード結合を有効にできます。

- 例に示したとおり、テーブルと coshard with キーワードとの構造的な関係を表す SASK。この例では、ORDER テーブル内の customerid が CUSTOMER テーブル内の RowId への参照となっています。
- RowID に対応しないため、VEHICLE テーブルおよび CITATION テーブルの等値の vin フィールドと vehiclenumber フィールドの使用で説明したとおりに coshard with を使用してコシャードできない、意味的に重要なフィールドを含む UDSK(多くの結合で使用されますが、表面的またはアドホックな関係を表すフィールドを含む UDSK は、通常あまり役立ちません)。

結合がないクエリやシャード・データとシャード化されていないデータを結合するクエリと同様に、コシャード結合は、シャード数の増加に伴う拡張性に優れているだけでなく、結合されるテーブルの数の増加に伴う拡張性にも優れています。コシャードされていない結合は、シャードや結合されるテーブルの数が適度であれば、優れたパフォーマンスを発揮しますが、それらの数の増加に伴う拡張性にはそれほど優れていません。こうした理由から、例えば、クエリが頻繁に実行されるフィールドのセットのパフォーマンスを向上させるためにインデックスを考慮するのと同様に、この段階でコシャード結合を慎重に検討する必要があります。

シャード・キーを選択する際は、これらの一般的な考慮事項に留意してください。

- シャード・テーブルのシャード・キーは変更できず、その値を更新することもできません。
- 他の条件がすべて同じであれば、シャード間でのテーブルの行のバランスの取れた分散はパフォーマンスを向上させ、行の分散に使用されるアルゴリズムは、シャード・キーにさまざまな値が大量に含まれるが、主だった異常値はない場合に、最適なバランスを提供します（頻度に関して）。これは既定の RowID が一般にうまく機能するためです。類似した特性を持つ適切な UDSK は効率的であることも多いですが、不適切な UDSK はパフォーマンスの大きな向上が見られない不均衡なデータ分散につながる可能性があります。
- 大きいテーブルがはるかに小さいものに頻繁に結合されるときに、大きいテーブルをシャード化し、小さいテーブルをシャード化されないようにすると、コシャード結合を有効にするより効果的な場合があります。

3.3.2.2 一意制約の評価

シャード・テーブルに一意制約がある場合（"InterSystems SQL リファレンス" の "CREATE TABLE" エントリにある "フィールド制約" および "複数フィールドでの一意制約" を参照）、すべてのシャード間で一意性が保証されます。これは通常、挿入または更新される各行について、すべてのシャード全体で一意性を強制する必要があるため、挿入/更新のパフォーマンスが大幅に低下することを意味します。ただし、シャード・キーが一意キーのフィールドのサブセットである場合は、行が挿入または更新されるシャードでローカルに一意性を強制することにより、すべてのシャード全体で一意性を保証できるため、このようなパフォーマンスへの影響が回避されます。

例えば、指定したキャンパスの OFFICES テーブルに buildingnumber フィールドと officenumber フィールドが含まれているとします。建物番号はキャンパス内で一意であり、オフィス番号は各建物内で一意です。この 2 つを組み合わせて、各従業員のオフィス・アドレスをキャンパス内で一意にする必要があります。したがって、テーブルに対する一意制約を以下のように配置します。

```
CREATE TABLE OFFICES (countrycode CHAR(3), buildingnumber INT not null, officenumber INT not null,
  employee INT not null, CONSTRAINT address UNIQUE (buildingname,officenumber))
```

ただし、テーブルをシャーディングする予定で、挿入/更新のパフォーマンスへの影響を回避するには、buildingnumber、officenumber、またはその両方をシャード・キーとして使用する必要があります。例えば、buildingnumber に対してシャーディングを行う（`shard key (buildingnumber)` を上記の文に追加することによる）場合、各建物のすべての行が同じシャードに配置され、アドレスが "building 10, office 27" である従業員の行を挿入する際に、buildingnumber=10 であるすべての行を含むシャードに、アドレスの一意性をローカルに適用できます。officenumber に対してシャーディングを行う場合、officenumber=27 であるすべての行が同じシャードに配置され、"building 10, office 27" の一意性を、そのシャードにローカルに適用できます。一方、SASK、または UDSK として employee を使用する場合は、buildingnumber と officenumber の任意の組み合わせが任意のシャードに現れる可能性があるため、"building 10, office 27" の一意性をすべてのシャードで適用しなければならず、パフォーマンスに影響が生じます。

このような理由により、次のいずれかが当てはまらない限り、シャード・テーブルでの一意制約の定義は避けることをお勧めします。

- すべての一意制約がサブセットとしてシャード・キーと共に定義されている（これは一般に、SASK や他の UDSK ほど効果的ではない）。
- 挿入および更新のパフォーマンスが、該当するテーブルのクエリ・パフォーマンスと比べてはるかに重要性が低いと思われる。

注釈　アプリケーション・コードで一意性を強制することにより (あるカウンタに基づくなど)、テーブル内の一意制約の必要性を排除でき、シャード・キーの選択が簡素化されます。

3.3.2.3 テーブルの作成

クラスタ内の任意のデータ・ノードのクラスタ・ネームスペースで標準の CREATE TABLE 文 (″InterSystems SQL リファレンス″ の ″CREATE TABLE″ を参照) を使用して空のシャード・テーブルを作成します。″シャード・キーの選択″ の例で示したように、テーブルを作成する際、2 種類のシャーディング仕様があります。

- システムが割り当てたシャード・キー (SASK) でシャーディングを行うには、CREATE TABLE 文に shard キーワードを含めます。
- ユーザ定義のシャード・キー (UDSK) でシャーディングを行うには、shard の後に key とシャーディングするフィールドを指定します (shard key (customerid, purchaseid) など)。

注釈　″CREATE TABLE″ リファレンス・ページの ″主キーの定義″ で説明しているように、テーブルを作成する際に PK_IS_IDKEY オプションを設定すると、テーブルの RowID が主キーになります。このような場合、既定のシャード・キーを使用することは、主キーがシャード・キーとなることを意味します。ただし、主キーをシャード・キーとして使用する場合は、テーブルを作成する前にこの設定の状態を確認しなくても済むように、シャード・キーを明示的に指定することをお勧めします。

ノード 1 または別のデータ・ノードの管理ポータルの [シャード構成] ページ (**[システム管理]→[構成]→[システム構成]→[シャード構成]**) に移動して、クラスタ・ネームスペースを選択し、**[シャードテーブル]** タブを選択することによって、名前、所有者、シャード・キーを含む、クラスタ上のすべてのシャード・テーブルのリストを表示できます。データをロードしたテーブルに対し、**[詳細]** リンクをクリックすると、クラスタ内の各データ・ノードに格納されているテーブルの行数を確認できます。

シャード・テーブルの作成における制約

以下の制約が、シャード・テーブルの作成に適用されます。

- ALTER TABLE を使用して既存のシャード化されていないテーブルをシャード・テーブルに入れることはできません (ただし、ALTER TABLE を使用してシャード・テーブルを変更することはできます)。
- SHARD KEY フィールドのデータ型は、数値または文字列でなければなりません。シャード・キー・フィールドについて現在サポートされている照合は、完全一致、SQLString、および SQLUpper のみで、切り捨ては行われません。
- ROWVERSION フィールドおよび SERIAL (%Counter) フィールド以外のすべてのデータ型がサポートされます。
- シャード・テーブルに %CLASSPARAMETER VERSIONPROPERTY を含めることはできません。

このセクションのトピックの詳細および例は、″InterSystems SQL リファレンス″ の ″CREATE TABLE″ を参照してください。

シャード・クラスを使用したシャード・テーブルの定義

DDL を使用したシャード・テーブルの定義に加え、シャード・クラス・キーワードを使用してクラスをシャード・クラスとして定義することができます。詳細は、″InterSystems SQL の使用法″ の “テーブルの定義” の章にある ″永続クラスの作成によるシャード・テーブルの定義″ を参照してください。クラス・コンパイラが拡張され、コンパイル時にシャーディングと互換性のないクラス定義機能 (カスタマイズされたストレージ定義など) の使用に対して警告を発行するようになりました。さらに成熟した作業負荷メカニズムや一部の互換性のない機能のサポートは、InterSystems IRIS の今後のバージョンで導入される予定です。

3.3.3 クラスタへのデータのロード

データは、SQL をサポートする InterSystems IRIS インタフェースを介してシャード・テーブルにロードできます。InterSystems IRIS JDBC ドライバに組み込まれている透過的な並列ロード機能では、シャード・テーブルへのデータの高速一括ロードがサポートされています。これらのオプションを、以下で説明します。

- JDBC クライアント・ツールや SQL シェルなど、SQL をサポートする任意の InterSystems IRIS インタフェースを介してクラスタ上の空のシャード・テーブルにデータをロードできます。十分に検証されたデータで迅速にテーブルを生成することを目的とした LOAD DATA コマンドは、テーブルまたはファイルからデータをロードします。INSERT SELECT FROM などの INSERT 文を使用してデータをロードすることもできます。
- InterSystems IRIS JDBC ドライバは、シャード・テーブルのロードに対する特定の最適化を実装します。シャード・テーブルへの INSERT 文を準備する際、JDBC クライアントは自動的に各データ・ノードへの直接接続を開き、これらに挿入された行を分配することで、特定の構成や API 呼び出しを行うことなく、大幅にパフォーマンスを向上させます。JDBC を使用するどのアプリケーションのシャード・テーブルのロードも、透過的にこの機能を利用します。

  注釈　ほとんどのデータ・ロード操作では、JDBC ドライバは、クラスタによって仲介されるデータ・ノードへの直接接続を使用します。これは、クラスタに割り当てる際に使用された IP アドレスまたはホスト名でデータ・ノードにアクセスすることをドライバ・クライアントに要求し、これが不可能な場合、このようなクエリを実行できないことを意味します。

3.3.4 シャード化されていないテーブルの作成とロード

シャード・テーブルと同様に、一般的な方法を使用して任意のデータ・ノードにシャード化されていないテーブルを作成し、それらのテーブルにデータを読み込むことができます。これらのテーブルは、シャード化されていないクエリとそれらをシャード・テーブルに結合するシャード・クエリの両方について、クラスタですぐに使用できます(これは、シャード化されていないテーブルを、それらを必要とする可能性がある各ノードに明示的に複製する必要があるアーキテクチャとは対照的です)。シャード化されていないものとしてロードするテーブルを選択する際のガイダンスについては、"シャーディングに関する既存のテーブルの評価" を参照してください。

3.4 シャード・クラスタにおけるクエリ

マスタ・ネームスペースとそこに格納されているシャード・テーブルは完全に透過的であり、マスタ・ネームスペース内のシャード・テーブルとシャード化されていないテーブルの任意の組み合わせを含む SQL クエリは、InterSystems IRIS ネームスペース内のテーブルに対する SQL クエリとまったく同じです。シャード・テーブルやシャード・キーを特定するための特殊なクエリ構文は必要ありません。クエリでは、複数のシャード・テーブルを結合することも、シャード・テーブルとシャード化されていないテーブルを結合することもできます。以下に示した InterSystems IRIS シャード・クラスタの初期バージョンにおける制限および制約以外のすべてがサポートされています。これらの制限および制約をすべて取り除くことを目標としています。

- 2 つのテーブルがコシャードされる場合、シャード・テーブルに強制される参照整合性制約は外部キーのみです。また、サポートされる唯一の参照アクションは NO ACTION です。
- シャード・キー・フィールドのデータ型は、数値または文字列でなければなりません。シャード・キー・フィールドについて現在サポートされている照合は、完全一致、SQLString、および SQLUpper のみで、切り捨ては行われません。
- シャード・テーブルの行レベルのセキュリティは、現在サポートされていません。
- SQL ゲートウェイ接続経由でコンテンツを取得しているリンク・テーブルをシャード化することはできません。
- 以下の InterSystems IRIS SQL 拡張の使用は、現在サポートされていません。

- %FOREACH および %AFTERHAVING を含む集約関数拡張
- 入れ子になった集約関数
- SELECT 節または HAVING 節に集約関数と集約していないフィールドの両方を使用したクエリ (この集約していないフィールドを、GROUP BY 節で GROUP 項目としても使用している場合を除く)
- FOR SOME %ELEMENT 述語条件
- %INORDER キーワード

注釈　データ・ノードでクエリ・キャッシュを明示的に削除する場合、すべてのクエリ・キャッシュをマスタ・ネームスペースから削除することも、特定のテーブルについてクエリ・キャッシュを削除することもできます。これらのどちらのアクションでも、データ・ノードに削除が伝播されます。個々のクエリ・キャッシュの削除がデータ・ノードに伝播されることはありません。

クエリからタイムアウト・エラーが返された場合や messages.log に ECP 接続の問題が報告された場合、その一般的な原因はネットワークの問題です。このような場合は、%SYSTEM.Sharding API の $SYSTEM.Sharding.VerifyShards() メソッドを呼び出すことをお勧めします。このメソッドによってテストが実行され、接続の回復が試行されて、その過程での診断情報が報告されます。

クエリで報告される特権エラーの原因は、さまざまなシャードにユーザが割り当てた特権の不一致にあることが考えられます。例えば、特定のテーブルやビューに SELECT 特権を割り当てている場合です。サーバごとに特権を検証し、その同期を図ることをお勧めします。IRIS の今後のバージョンでは、このプロセスを自動化する予定です。

3.5 その他のシャード・クラスタ・オプション

シャーディングには、ニーズに適したさまざまな構成およびオプションが用意されています。このセクションでは、関連するその他のオプションについて簡単に説明します。内容は以下のとおりです。

- データ・ノードの追加とデータの再分散
- ミラー・データ・ノードによる高可用性
- 作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入
- 1 つのシステムへの複数のデータ・ノードのインストール

クラスタにおけるこれらのオプションの効果を評価する際には、インターシステムズのサポート窓口までお問い合わせください。

3.5.1 データ・ノードの追加とデータの再分散

InterSystems IRIS のシャーディングは、スケーラビリティと弾力性を実現するために設計されています。"InterSystems IRIS シャード・クラスタの計画" で説明しているように、最初の導入時にクラスタに含めるデータ・ノードの数は、シャード・テーブルで予測される作業セット、利用可能な計算リソースを含む (ただし、これらに限定されない) いくつかの要素の影響を受けます。ただし、時間の経過に伴って、クラスタ上のシャード・データの量が大幅に増加したり、リソース制約が削除されるといったさまざまな理由により、データ・ノードを追加する場合があります。"シャード・クラスタの導入" で説明されている自動または手動の導入方法を使用して、データ・ノードを追加できます。

"InterSystems IRIS のシャーディングの概要" で説明しているように、シャーディングでは、すべてのデータ・ノードでクエリの分解とそれらの実行を並列で実行することにより、クエリ処理のスループットを調整し、結果はマージおよび集約され、完全なクエリ結果としてアプリケーションに返されます。一般に、データ・ノードに格納されるシャード・データの量が

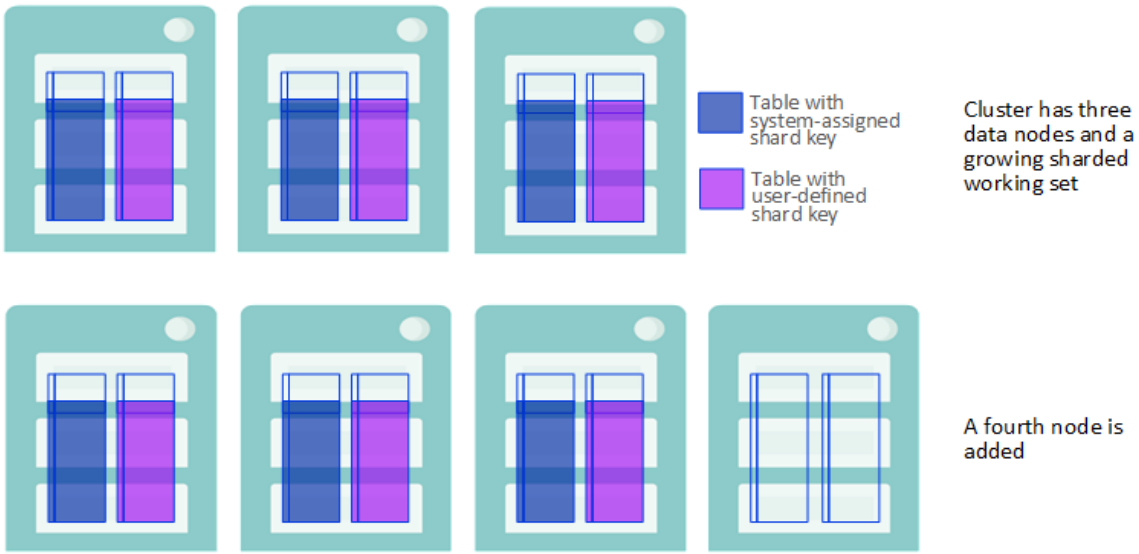
多いほど、結果を返すのに時間がかかり、全体的なクエリのパフォーマンスは、最も遅いデータ・ノードによって制限されます。したがって、最適なパフォーマンスを得るには、シャード・データの格納が、クラスタのデータ・ノード間でほぼ均等になるようにする必要があります。

データ・ノードの追加後は該当しませんが、すべてのデータ・ノードでクラスタのシャード・データを再分散することで、このほぼ均等な分散がリストアされます。クラスタはその動作を中断せずに再分散できます。

データ・ノードの追加およびデータの再分散のプロセスを、次の例で説明します。

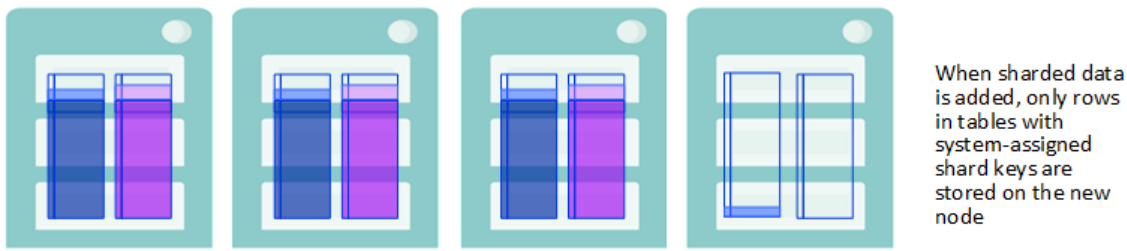
- データ・ノードをクラスタに追加した後、既存のシャード・テーブルの行は、再分散するまで元のノードでそれらが配置されていた場所に維持されます。

図 3-2: データ・ノードが追加される



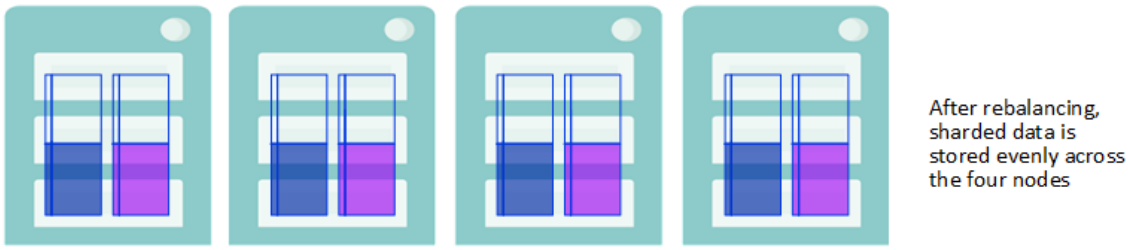
- 既存のテーブルに追加した行、または新しく作成してデータを読み込んだテーブルの形式でスケーリングされたクラスタにシャード・データを追加した場合、そのストレージは、各テーブルのシャード・キーによって異なります。
  - テーブルにシステム割り当てシャード・キー (SASK) がある場合、新しい行は、新しいノードを含むすべてのデータ・ノードに均等に格納されます。
  - テーブルにユーザ定義のシャード・キー (UDSK) がある場合、新しい行は元のデータ・ノード・セットにのみ均等に格納され、新しく追加したノードには格納されません。(ただし、新しいノードが追加される前に、既存の UDSK テーブルが存在しなかった場合は、新しい UDSK テーブル内の行がすべてのデータ・ノードに分散されます。)

図 3-3: 新しいデータがシャード・キーに基づいて格納される



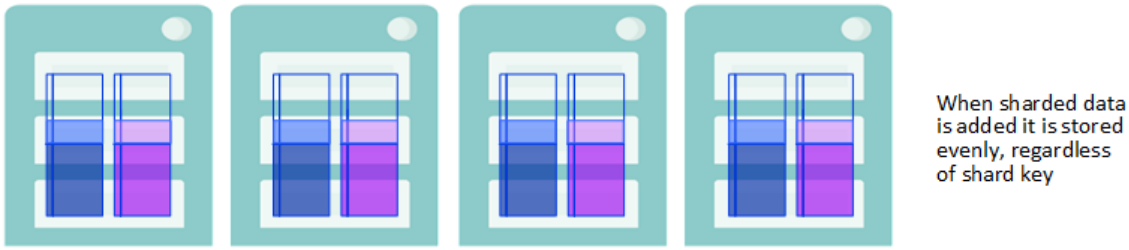
- データ・ノードの追加後、クラスタ上でのクエリの並列処理を最大限に活用するには、クラスタに格納されたシャード・データを再分散し、今後クラスタに追加されるデータの分散された格納を可能にします。

図 3-4: 再分散によりシャード・データが均等に格納される



- 再分散後、両方の行が既存のテーブルに追加され、新しく作成されたテーブルの行も、シャード・キーに関係なくすべてのデータ・ノードに分散されます。したがって、再分散後、すべてのシャード・データ(既存のテーブル、既存のテーブルに追加された行、新しいテーブル内) は、すべてのデータ・ノードに均等に分散されます。

図 3-5: 新しいシャード・データが均等に格納される



再分散操作は以下の 2 つの方法のいずれかで開始できます。

- 管理ポータルの [Configure Node-Level] ページ (**[システム管理]→[構成]→[システム構成]→[Sharding]→[Configure Node-Level]**) で **[Rebalance]** をクリックすることにより表示される [Rebalance] ダイアログを使用する。
- $SYSTEM.Sharding.Rebalance() API 呼び出しを使用する。

$SYSTEM.Sharding.Rebalance() のクラス・リファレンス・ドキュメントでは、いずれかのインタフェースを使用して指定できるパラメータについて説明しているほか、データ・ノード間でのシャード・データの再分散の詳細な手法についても説明しています。

再分散操作では、シャード・テーブル上でのほぼすべての操作が許可されていますが、シャード・テーブルへの JDBC バッチ挿入 (または仲介される JDBC バッチ挿入を使用するすべてのバルク・ロード・ユーティリティ) は例外で、これが試みられるとエラーが返されます。再分散操作の実行中には、クエリ・パフォーマンスに小さな悪影響を及ぼす場合があり、さらに小さな規模では、更新、挿入、削除などを含むその他の操作、テーブルの作成、変更、および削除、新しいシャードの割り当てにも悪影響を及ぼす場合があります。

パフォーマンスがかなりの懸念事項となる場合は、その操作に時間制限を設け、トラフィックが少ない時間やメンテナンスの時間にスケジュールすることもできます。時間制限に達すると、再分散操作は (まだ完了していない場合) データの移動を停止します (ただし、一部のクリーンアップ・タスクは短時間継続する場合があります)。再分散するデータが残っている場合は、選択した時間に、前の再分散が中断された場所を取得して別の再分散操作を開始できます。このアプローチを使用して、ニーズに合った長さでスケジュールされた一連の操作により、クラスタを完全に再分散できます。

この API 呼び出しを使用する場合、呼び出しにより移動されるデータの最小量を指定することもできます。指定された制限時間内にそれだけの量のデータを移動できない場合、再配分は発生しません。データ・ノードを追加したら、そのデータを再分散することによりパフォーマンスを最大限に高めることに留意してください。

3.5.2 ミラー・データ・ノードによる高可用性

InterSystems IRIS ミラーとは、物理的に独立した複数の InterSystems IRIS インスタンスの論理グループで、プロダクション・データベースの同一コピーを同時に保持します。これによって、データベースへのアクセスを提供しているインスタン

スが使用不可になった場合でも、別のインスタンスが自動的にすばやく引き継ぐことができます。この自動フェイルオーバー機能により、ミラー内の InterSystems IRIS データベースに高可用性が提供されます。InterSystems IRIS のミラーリングの詳細は、"高可用性ガイド" を参照してください。

シャード・クラスタ内のデータ・ノードをミラーリングすると、データ・ノードにフェイルオーバー機能を提供し、高可用性を実現できます。シャード・クラスタ内のミラーリングされた各データ・ノードには、少なくともインスタンスのフェイルオーバー・ペアが含まれます。ペアの一方は常にプライマリとして動作するのに対し、他方はバックアップとして動作し、フェイルオーバー・パートナーが利用できなくなった場合にプライマリとして引き継ぐことができます。シャード・クラスタ内のデータ・ノード・ミラーに DR (災害復旧) 非同期メンバを 1 つ以上含めることもできます。このメンバをフェイルオーバー・メンバに昇格させることで、無効になったフェイルオーバー・パートナーを置き換えたり、フェイルオーバー・パートナーが両方とも利用できなくなった場合に災害復旧を提供したりできます。一般的な構成では、DR 非同期をフェイルオーバー・ペアとは別のデータ・センターまたはクラウドのアベイラビリティ・ゾーンに配置して、すべてのメンバが同じ障害や停止の影響を受ける可能性を最小限に抑えることを強くお勧めします。

シャード・クラスタは、すべてミラーリングするか、まったくミラーリングしないかのどちらかにする必要があります。同じクラスタ内にミラーリングされたデータ・ノードとミラーリングされていないデータ・ノードを混在させることはできません。

(管理ポータルまたは %SYSTEM.Cluster API のいずれかを使用した) ミラーリングされたデータ・ノードの手動での構成手順では、既存のミラー構成が認識されます。つまり、既存の環境と要件に応じて、ミラーリングされていないインスタンスか、ミラーリングされたインスタンスのいずれかから、以下のようにミラーリングされたデータ・ノードのシャード・クラスタを構成できます。

- ミラーリングされたシャード・クラスタとしてミラーリングされていないインスタンスを構成する場合、追加する対象の各プライマリが、クラスタにアタッチされる前に自動的にミラー・プライマリとして構成されます。その後、別のミラーリングされていないインスタンスを、そのバックアップとして追加できます。追加したインスタンスは、自動的に適切に構成されてからクラスタにアタッチされます。
- 既存のミラーをシャード・クラスタとして構成する場合は、追加する各プライマリは、既存のミラー構成に変更を加えることなく、データ・ノード・プライマリとしてクラスタにアタッチされます。その後、フェイルオーバー・パートナーをそのプライマリのバックアップとして指定して、クラスタに追加できます。(データ・ノード・プライマリとして追加したミラーリングされたインスタンスにフェイルオーバー・パートナーがない場合は、ミラーリングされていないインスタンスをバックアップとして指定できます。これは自動的に適切に構成されてからクラスタにアタッチされます。)
- 同様に、ミラーリングされていないインスタンスを、データ・ノード・ミラーの DR 非同期メンバとして追加し、自動的に適切に構成してからクラスタにアタッチするか、フェイルオーバー・メンバが既にクラスタにアタッチされている既存のミラーのDR 非同期メンバを、適切に指定することで追加することができます。
- どの場合も、クラスタ・ネームスペースとマスタ・ネームスペース (既定では **IRISCLUSTER** と **IRISDM**) のグローバル・データベースは、ミラー (前者はすべてのデータ・ノード、後者はノード 1 のミラー) に追加されます。既存のミラー・メンバを構成する場合は、ミラーリングされたデータベースは、シャード・クラスタの構成に従ってミラーリングされたまま残ります。

一般には、ミラーリングされていないすべてのインスタンスで開始してシャード・クラスタ導入の一環としてミラーを構成するか、既存のミラーからすべてのデータ・ノードを構成するのがベスト・プラクティスです。

"シャード・クラスタの導入" で説明されているいずれかの方法を使用して、ミラーリングされたシャード・クラスタを導入できます。そこで説明されている自動導入方法にはすべて、ミラーリングがオプションとして含まれます。このセクションでは、ミラーリングされたクラスタに関する手順を説明し、"シャード・クラスタの導入" のセクションに含まれる手動手順の最後のステップを置き換えます。まず、"シャード・クラスタの導入" のセクションで説明されているステップを実行してから、以下のように、ここで説明する手順の 1 つを使用して導入を完了します。

1. データ・ノードの計画
2. データベース・キャッシュとデータベースのサイズの見積もり
3. インフラストラクチャのプロビジョニングまたは特定

注釈　最初のステップで計画したミラーリングされたデータ・ノードにホストをそれぞれ 2 つ指定する必要があることに注意してください。例えば、8 個のデータ・ノードが必要な計画の場合、クラスタには 16 個のホストが必要です。シャード・クラスタ内のすべてのデータ・ノードに対して推奨されているように、ミラーを構成する 2 つのホストの仕様とリソースは、同じであるか、少なくともほぼ同等である必要があります。

4. データ・ノード・ホストへの InterSystems IRIS の導入
5. 以下のいずれかを使用した、ミラーリングされたクラスタ・ノードの構成
   - 管理ポータル
   - %SYSTEM.Cluster API

ただし、クラスタを導入する際のベスト・プラクティスとして、以下を推奨します。

- クラスタ内のミラーリングされたすべてのデータ・ノード間でアプリケーション接続を負荷分散します。
- クラスタにデータを保存する前に、ミラーリングされたシャード・クラスタを導入して完全に構成し、ミラーリングされたデータ・ノードのすべてのメンバ (フェイルオーバー・ペアとすべての DR 非同期) をクラスタにアタッチします。ただし、データが存在するかどうかに関係なく、既存のミラーリングされていないクラスタをミラーリングされたクラスタに変換することもできます。
- データ・ノード・ミラーのフェイルオーバー後にクエリを透過的に実行できるようにするため、ミラーリングされたクラスタに計算ノードを含めます。

注釈　管理ポータルのミラーリングのページおよび %SYSTEM.MIRROR API では、ここで説明されている %SYSTEM.Cluster API および管理ポータルのシャーディングのページより多くのミラー設定を指定できます。″高可用性ガイド″ の “ミラーリングの構成” の章を参照してください。ミラーリングされていないインスタンスからクラスタを作成し、管理ポータルまたは API で自動的にミラーを構成する予定であっても、作業の前に ″“ミラーリングの構成”″ の章の ″ミラーの作成″ で手順と設定を確認することをお勧めします。

このセクションの手順を使用して、ミラーリングされたネームスペース・レベルのクラスタを導入することはできません。ただし、ミラーリングされていないネームスペース・レベルのクラスタを導入してから (″ネームスペース・レベル・アーキテクチャの導入″ を参照)、そのクラスタをミラーリングされたクラスタに変換できます (″ミラーリングされていないクラスタからミラーリングされたクラスタへの変換″ を参照)。

3.5.2.1 ミラーリングされたクラスタの考慮事項

ミラーリングされたシャード・クラスタの導入時には、以下のセクションで説明する重要なポイントに留意してください。

ミラーリングされたクラスタに計算ノードを含めることによる、フェイルオーバー前後での透過的なクエリ実行

計算ノードには永続データが保存されないため、計算ノードそのものはミラーリングしませんが、ミラーリングされたクラスタに計算ノードを含めると便利な場合があります。これは、関係のあるワークロードが、″作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入″ (計算ノードとそれらの導入手順の詳細情報を提供) で説明されている高度なユース・ケースに一致しない場合でも同様です。

ミラーリングされたシャード・クラスタが非同期クエリ・モード (既定) である場合に、シャード・クエリの実行中にデータ・ノード・ミラーがフェイルオーバーすると、エラーが返され、アプリケーションはクエリを再試行しなければなりません。この問題に対処する方法は 2 つあります。つまり、以下のように、フェイルオーバー前後でシャード・クエリを透過的に実行できるようにすることです。

- クラスタを同期クエリ・モードに設定する。ただし、これには欠点があります。同期モードの場合、シャード・クエリをキャンセルできず、**IRISTEMP** データベースの使用量が増えます。このため、シャード・クエリが増加すると、利用可能なストレージ領域がすべて消費されて、クラスタの動作が中断するリスクが高くなります。
- 計算ノードをクラスタに含める。計算ノードには、計算ノードが割り当てられているミラーリングされたデータ・ノードとのミラー接続があるため、フェイルオーバーの前後で非同期モードでクエリを透過的に実行できます。

上記のオプションを考慮すると、ワークロードにとってフェイルオーバー前後でクエリを透過的に実行できることが重要である場合は、ミラーリングされたシャード・クラスタに計算ノードを含めることをお勧めします（少なくとも、ミラーリングされたデータ・ノードと同じ数の計算ノードが必要です）。計算ノードを含めることができない状況の場合は、$SYSTEM.Sharding.SetOption() API 呼び出しの RunQueriesAsync オプション（%SYSTEM.Sharding API を参照）を使用して、クラスタを非同期モードに変更できます。ただし、変更するのは、シャード・クエリをキャンセルできること、および **IRISTEMP** のサイズを管理できることよりも、フェイルオーバー前後でクエリを透過的に実行できることの方が重要な場合だけにしてください。

導入前のクラスタおよびマスタ・ネームスペースの作成

シャード・クラスタは最初のデータ・ノードを構成するときに初期化されます。最初のデータ・ノードをデータ・ノード 1、または単にノード 1 と呼びます。ここには既定で、クラスタとマスタ・ネームスペース（それぞれ **IRISCLUSTER** および **IRISDM** という名前）と共に、その既定のグローバル・データベースと必要なマッピングの作成が含まれます。ただし、クラスタ・ネームスペースとマスタ・ネームスペースの名前、またはそのグローバル・データベースの特性（あるいはその両方）を制御するため、クラスタを構成する前に一方または両方のネームスペースと既定のデータベースを作成しておき、この手順でそれらを指定できます。ミラーリングされたシャード・クラスタの導入時にこれを行うことを計画している場合、ミラーリングされていないインスタンスで開始することはできませんが、代わりに次の手順を示された順序で実行する必要があります。

1. それぞれ 2 つのフェイルオーバー・メンバを含め、想定される各データ・ノードにミラーを構成します。
2. 各ミラー・プライマリ上で対象のクラスタ・ネームスペースを作成し（既定の名前 **IRISCLUSTER** またはオプションで別の名前を使用）、データ・ノード 1 としてアタッチするミラーのプライマリ上で対象のマスタ・ネームスペース（既定の名前 **IRISDM**）を作成します。各ネームスペースの作成時に、[グローバルのための既存のデータベースを選択] のプロンプトで [新規データベース作成] を選択し、データベース・ウィザードの 2 ページ目の下部にある [ミラー・データベース?] に [はい] と設定して、ミラーリングされたデータベースとしてネームスペースの既定のグローバル・データベースを作成します。
3. このセクションで説明したとおり、データ・ノードとしてミラーを構成し、作成したネームスペースをクラスタおよびマスタ・ネームスペースとして指定します。

すべてのミラー・メンバでの IP アドレスのオーバーライドの有効化

場合によっては、対象のクラスタ・ノード上の InterSystems IRIS インスタンスに認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このため、または他の何らかの理由により、他のクラスタ・ノードが代わりにその IP アドレスを使用してノードと通信する場合、管理ポータルの [Override hostname with IP address] プロンプトで、または $SYSTEM.Cluster.InitializeMirrored() および $SYSTEM.Cluster.AttachAsMirroredNode() 呼び出しの引数として、そのノードの IP アドレスを指定することでこれを有効にできます。ミラーの 1 つのメンバに対してこれを実行したら、以下のようにすべてのミラー・メンバに対してこれを実行する必要があります。

- ミラーリングされていないインスタンスをミラーリングされたデータ・ノードとして構成する場合は、上記のプロンプトまたは呼び出しを使用して、すべてのミラー・メンバに対して IP アドレスのオーバーライドを必ず有効にしてください。
- ただし、既存のミラーをデータ・ノードとして構成する場合は、クラスタに追加する前に、ミラーのすべてのメンバに対して IP アドレスのオーバーライドを有効にする必要があります。管理ポータルと API のどちらの手順を使用しているかに関係なく、これは、各ノードで、InterSystems ターミナル・ウィンドウを開き、（任意のネームスペースで）$SYSTEM.Sharding.SetNodeIPAddress()（"%SYSTEM.Sharding API" を参照）を呼び出すことによって行ってください。以下に例を示します。

  ```
  set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
  ```

  ノードでこの呼び出しを使用したら、他の API 呼び出しでそのノードを参照するのに、ホスト名ではなく、指定した IP アドレスを使用する必要があります（$SYSTEM.Cluster.AttachAsMirroredNode() 呼び出し（ここでアタッチされるプライマリを指定する必要があります）を使用してミラー・バックアップをクラスタにアタッチする場合など）。

ミラーリングされたクラスタのメタデータの更新

%SYSTEM.Cluster API および管理ポータルのシャーディングのページ以外で、例えば管理ポータルのミラーリングのページや SYS.Mirror API を使用して、クラスタのミラーリングされたデータ・ノードを変更する場合は、変更後にミラーリングされたクラスタのメタデータを更新する必要があります。詳細は、"変更をミラーリングするためのクラスタ・メタデータの更新" を参照してください。

3.5.2.2 管理ポータルを使用したミラーリングされたクラスタの構成

ブラウザで管理ポータルを開く方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎：IDE の接続" の "InterSystems IRIS 接続情報" のセクションで、キットからインストールしたインスタンスの手順を参照してください。

ミラーリングされたクラスタを管理ポータルを使用して構成するには、以下の手順を実行します。

1. 対象のノード 1 プライマリと対象のノード 1 バックアップの両方でインスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→[Sharding]→[シャード有効]** の順に選択し、表示されるダイアログで [OK] をクリックします (ここで **[ECP 接続の最大数]** 設定を変更する必要はありません)。続いて、指示に従ってインスタンスを再起動します。管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動したら再読み込みするだけでかまいません。

2. 対象のプライマリで、[Configure Node-Level] ページ (**[システム管理]→[構成]→[システム構成]→[Sharding]→[Configure Node-Level]**) に移動して、**[構成]** ボタンをクリックします。

3. [Configure Node-Level Cluster] ダイアログで **[Initialize a new sharded cluster on this instance]** を選択し、表示されるプロンプトに次のように応答します。

   a. **[Cluster namespace]** および **[マスターネームスペース]** のドロップダウンからそれぞれクラスタ・ネームスペースとマスタ・ネームスペースを選択します。どちらのドロップダウンにも以下が含まれます。

      - 作成する新しいネームスペースの既定の名前 (**IRISCLUSTER** および **IRISDM**) と、その既定のグローバル・データベース。
      - 対象となるすべての既存のネームスペース。

      シャード・クラスタを初期化すると、既定で、クラスタとマスタ・ネームスペース (それぞれ **IRISCLUSTER** および **IRISDM** という名前) と共に、その既定のグローバル・データベースと必要なマッピングが作成されます。ただし、クラスタ・ネームスペースとマスタ・ネームスペースの名前、およびそのグローバル・データベースの特性を制御するため、クラスタを構成する前に一方または両方のネームスペースと既定のデータベースを作成しておき、この手順でそれらを指定できます。例えば、"グローバル・データベース・サイズ" で説明されている考慮事項を踏まえ、クラスタ・ネームスペースの既定のグローバル・データベースの特性や、クラスタ内のすべてのデータ・ノードに複製されるシャード・データベースを制御するために、この方法を実行できます。

      注釈　シャード・クラスタとして既存のミラーを構成する際に既存のネームスペースを使用する場合は、"導入前のクラスタおよびマスタ・ネームスペースの作成" の手順に従います。

      クラスタ・ネームスペースとして指定した既存ネームスペースの既定グローバル・データベースになんらかのグローバルまたはルーチンがあると、初期化がエラーで失敗します。

   b. 場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、**ホスト名のオーバーライド**のプロンプトで IP アドレスを入力します。

   c. [Enable mirroring] を選択し、アービターを構成する場合はその場所とポートを追加します (これは強く推奨されるベスト・プラクティスです)。

4. [OK] をクリックして [Configure Node-Level] ページに戻ります。2 つのタブ **[シャード]** と **[シャードテーブル]** が表示されています。ノード 1 がクラスタ・アドレス (次の手順で必要になります) を含めて **[シャード]** に表示されるので、参

照用にノード 1 の管理ポータルで [Configure Node-Level] ページを開いたままにしておくことができます。まだバックアップを追加していないため、ミラー名はまだ表示されません。

5. 対象のノード 1 バックアップで、プライマリの [Configure Node-Level] ページに移動し、**[構成]** ボタンをクリックします。
6. [Configure Node-Level Cluster] ダイアログで **[Add this instance to an existing sharded cluster]** を選択し、表示されるプロンプトに次のように応答します。
   a. [Configure Node-Level] ページの **[シャード]** タブに、**[Cluster URL]** として、ノード 1 プライマリに対して表示されているアドレスを入力します (前のステップを参照)。
   b. **[ロール]** プロンプトで **[データ]** を選択して、インスタンスをデータ・ノードとして構成します。
   c. 場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、**ホスト名のオーバーライド**のプロンプトで IP アドレスを入力します。
   d. **[Mirrored cluster]** を選択して以下を実行します。
      - **[Mirror role]** ドロップダウンから **[バックアップ・フェイルオーバー]** を選択します。
      - 前のステップでノード 1 を初期化する際にアービターを構成した場合は、その際に追加したものと同じアービターの場所とポートを追加します。
7. **[OK]** をクリックして [Configure Node-Level] ページに戻ります。2 つのタブ **[シャード]** と **[シャードテーブル]** が表示されています。これまでに構成したノード 1 プライマリとバックアップは **[シャード]** のノード 1 の位置に、割り当てられているミラー名が含まれた状態で一覧表示されます。
8. 残りのミラーリングされたデータ・ノードに対し、**[シャード有効]** オプションの設定と両方のインスタンスの再起動から始めて、前のステップを繰り返します。プライマリをクラスタに追加する場合は、前述のように、ノード 1 プライマリのアドレスを **[Cluster URL]** として入力します。ただし、バックアップを追加する場合には、先ほど追加したプライマリのアドレスを **[Cluster URL]** として入力します (ノード 1 プライマリのアドレスではありません)。
9. ミラーリングされたデータ・ノードをそれぞれ追加したら、クラスタ内のプライマリの 1 つで [Configure Node-Level] ページに移動して **[シャードを検証]** をクリックし、新しいミラーリングされたノードが正しく構成されていて、他のノードと通信できることを検証します。この作業は、ミラーリングされたデータ・ノードをすべて追加するまで待つこともできます。また、構成するデータ・ノードが多数ある場合は、**[詳細設定]** ボタンをクリックし、[詳細設定] ダイアログで **[割り当て時に自動的にシャードを検証]** を選択することにより、検証操作を自動化できます(シャード・クラスタを導入する際は、このダイアログのその他の設定は既定値のままにします)。

3.5.2.3 %SYSTEM.Cluster API を使用したミラーリングされたクラスタ・ノードの構成

この API を使用して、ミラーリングされたクラスタを構成するには、以下の手順を実行します。

1. 対象のノード 1 プライマリで、インスタンスの InterSystems ターミナルを開き、$SYSTEM.Cluster.InitializeMirrored() メソッドを呼び出します。以下に例を示します。

```
set status = $SYSTEM.Cluster.InitializeMirrored()
```

注釈　これらの手順で詳述されている各 API 呼び出しの戻り値 (成功の場合は 1 など) を確認するには、以下を入力します。

```
zw status
```

呼び出しが成功しなかった場合、以下を入力することにより、わかりやすいエラー・メッセージが表示されます。

```
do $SYSTEM.Status.DisplayError(status)
```

“%SYSTEM.Cluster API を使用したクラスタの導入” の ″ノード 1 の構成″ で説明したように、この呼び出しは、$SYSTEM.Cluster.Initialize() と同じようにノード上のクラスタを初期化します。InitializeMirrored() の最初の 4 つの引数 (必須のものはありません) の説明については、このセクションを確認してください。これらの引数は Initialize() の引数と同じです。インスタンスがまだミラー・プライマリでない場合、次の 5 つの引数を使用してこれを構成します。既にプライマリである場合、これらは無視されます。ミラー引数は以下のとおりです。

- アービターのホスト
- アービターのポート
- 認証機関の証明書、ローカルな証明書、必要に応じて TLS でミラーを保護するために必要な秘密鍵ファイルを含むディレクトリ。呼び出しでは、これらのファイルがそれぞれ **CAFile.pem**、**CertificateFile.pem**、および **PrivateKeyFile.pem** という名前であると想定しています。
- ミラーの名前。
- このミラー・メンバの名前。

注釈　InitializeMirrored() 呼び出しは、以下の場合、エラーを返します。

- 現在の InterSystems IRIS インスタンスが既にシャード・クラスタのノードである場合。
- 現在のインスタンスが既にミラー・メンバであるが、プライマリではない場合。
- (最初の 2 つの引数で) 既に存在するクラスタ・ネームスペースまたはマスタ・ネームスペースを指定し、そのグローバル・データベースがミラーリングされていない場合。

2. 対象のノード 1 バックアップで InterSystems IRIS インスタンスのターミナルを開き、最初の引数のクラスタ URL としてノード 1 プライマリのホストとスーパーサーバ・ポート、2 番目の引数にミラー・ロール **backup** を指定して、$SYSTEM.Cluster.AttachAsMirroredNode() を呼び出します。以下に例を示します。

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://node1prim:1972","backup")
```

ノード 1 プライマリの初期化の際、InitializeMirrored() の 4 番目の引数として IP アドレスを指定した場合は、最初の引数でノード 1 を識別するためにホスト名の代わりに IP アドレスを使用します。以下に例を示します。

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://100.00.0.01:1972","backup")
```

注釈　そのホスト上にしかないコンテナ化されていない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。インスタンスのスーパーサーバ・ポート番号を表示または設定するには、そのインスタンスの管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します (インスタンスの管理ポータルを開いてスーパーサーバ・ポートを確認する方法の詳細は、コンテナに導入されたインスタンスの手順、または ″InterSystems IRIS の基礎 : IDE の接続″ の ″InterSystems IRIS 接続情報″ で、キットからインストールしたインスタンスの手順を参照してください)。

“%SYSTEM.Cluster API を使用したクラスタの導入” の ″残りのデータ・ノードの構成″ で説明しているように、この呼び出しにより、$SYSTEM.Cluster.AttachAsDataNode() と同様、ノードがデータ・ノードとしてアタッチされ、これがノード 1 ミラーのバックアップ・メンバとなることが保証されます。この呼び出しを発行する前に、ノードがノード 1 プライマリのバックアップとなっている場合 (つまり、既存のミラーをノード 1 として初期化している場合)、ミラー構成は変わりません。これがミラー・メンバではない場合は、ノード 1 プライマリのミラーにバックアップとして追加されます。どちらの場合も、ノード 1 プライマリのネームスペース、データベース、およびマッピング構成は、このノードに複製されます。(AttachAsMirroredNode の 3 番目の引数は、AttachAsDataNode と同じです。つまり、他のクラスタ・メンバがこのノードとの通信にホストの IP アドレスを使用する場合、このアドレスを含めます。)

ノード 1 ミラーの対象の DR 非同期メンバがある場合は、AttachAsMirroredNode() を使用してメンバをアタッチします。その際、以下の例のように、2 番目の引数として **backup** の代わりに **drasync** を使用します。

```
set status = $SYSTEM.Cluster.AttachAsMirroredNode("IRIS://node1prim:1972","drasync")
```

バックアップをアタッチするときと同じように、ミラーの既存のメンバをアタッチする場合、そのミラー構成が変更されるのではなく、必要なミラー構成が追加されます。どちらの場合も、ノード 1 プライマリのネームスペース、データベース、およびマッピング構成は、新しいノードに複製されます。

注釈　ノード 1 プライマリのミラーとは異なるミラーのメンバであるインスタンスをアタッチしようとすると、エラーが発生します。

3. ノード 1 以外のミラーリングされたデータ・ノードを構成するには、以下のように、$SYSTEM.Cluster.AttachAsMirroredNode() を使用してフェイルオーバー・ペアと DR 非同期の両方をクラスタにアタッチします。

   a. プライマリを追加する際には、既存のプライマリをクラスタ URL に指定し、2 番目の引数として primary を指定します。インスタンスがまだミラーのプライマリではない場合、4 番目の引数と続く 4 つの引数を使用してこれを新しいミラーの最初のメンバとして構成します。引数は、前述の InitializeMirrored() 呼び出しでリストされているとおりです。インスタンスが既にミラー・プライマリの場合、ミラー引数を指定しても無視されます。

   b. バックアップを追加する際には、対象のプライマリをクラスタ URL に指定し、2 番目の引数として backup を指定します。指定したノードがプライマリであるミラーに、バックアップとして既にインスタンスが構成されている場合、そのミラー構成は変わりません。インスタンスがまだミラー・メンバではない場合、これは 2 番目のフェイルオーバー・メンバとして構成されます。

   c. DR 非同期を追加する際には、対象のプライマリをクラスタ URL に指定し、2 番目の引数として drasync を指定します。指定したノードがプライマリであるミラーに、DR 非同期として既にインスタンスが構成されている場合、そのミラー構成は変わりません。インスタンスがまだミラー・メンバではない場合、これは DR 非同期として構成されます。

   注釈　AttachAsMirroredNode() 呼び出しは、以下の場合、エラーを返します。

   - 現在の InterSystems IRIS インスタンスが既にシャード・クラスタのノードである場合。
   - primary のロールが指定され、クラスタ URL (最初の引数) で指定されたクラスタ・ノードがミラー・プライマリではないか、現在のインスタンスがプライマリ以外のロールのミラーに属している場合。
   - backup のロールが指定され、最初の引数で指定されたクラスタ・ノードがミラー・プライマリではないか、バックアップ・フェイルオーバー・メンバを既に持つミラーのプライマリである場合。
   - drasync のロールが指定され、最初の引数で指定されたクラスタ・ノードがミラー・プライマリではない場合。
   - backup または drasync のロールが指定され、追加するインスタンスが既に、指定したプライマリを持つミラー以外のミラーに属している場合。
   - クラスタ・ネームスペース (または、ノード 1 バックアップを追加している場合はマスタ・ネームスペース) が現在のインスタンスに既に存在し、そのグローバル・データベースがミラーリングされていない場合。

4. すべてのデータ・ノードを構成したら、$SYSTEM.Cluster.ListNodes() メソッドを呼び出してこれらをリストできます。クラスタがミラーリングされている場合、このリストはミラーリングされているデータ・ノードの各メンバのミラー名とロールを示します。以下に例を示します。

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    Host          Port   Mirror  Role
1       Data        node1prim     1972   MIRROR1 Primary
1       Data        node1back     1972   MIRROR1 Backup
1       Data        node1dr       1972   MIRROR2 DRasync
2       Data        node2prim     1972   MIRROR2 Primary
2       Data        node2back     1972   MIRROR2 Backup
2       Data        node2dr       1972   MIRROR2 DRasync
```

3.5.2.4 ミラーリングされていないクラスタからミラーリングされたクラスタへの変換

ここでは、既存のミラーリングされていないシャード・クラスタをミラーリングされたクラスタに変換する手順を説明します。以下に、関連するタスクの概要を示します。

- 少なくとも十分な数の新しいノードをプロビジョニングして準備し、クラスタ内の既存のデータ・ノードそれぞれにバックアップを提供します。
- 既存のデータ・ノードそれぞれにミラーを作成してから、ノード 1 で $SYSTEM.Sharding.AddDatabasesToMirrors() を呼び出して、クラスタをミラーリングされた構成に自動的に変換します。
- 既存のデータ・ノード上に、ミラーリングされていないマスタ・データベースとシャード・データベースの調整されたバックアップを作成します (各ミラーの最初のフェイルオーバー・メンバ)。"シャード・クラスタの調整されたバックアップとリストア" を参照してください。
- 対象の 2 番目のフェイルオーバー・メンバ (新しいノード) それぞれに対して、参加させる最初のフェイルオーバー・メンバ (既存のデータ・ノード) を選択してから、最初のフェイルオーバー・メンバ上のミラーリングされたデータベースに対応するデータベースを新しいノード上に作成します。続いて、そのミラーに新しいノードを 2 番目のフェイルオーバー・メンバとして追加し、最初のフェイルオーバー・メンバ上に作成したバックアップからデータベースをリストアして、そのデータベースを自動的にミラーに追加します。
- データ・ノード・ミラー内に作成したフェイルオーバー・ペアに DR 非同期を追加するには、最初のフェイルオーバー・メンバ上のミラーリングされたデータベースに対応する新しいノード上にデータベースを作成します。続いて、そのミラーに新しいノードを DR 非同期として追加し、最初のフェイルオーバー・メンバ上に作成したバックアップからデータベースをリストアして、そのデータベースを自動的にミラーに追加します。
- 任意のミラー・プライマリ (元のデータ・ノード) で $SYSTEM.Sharding.VerifyShards() を呼び出して、バックアップに関する情報を検証し、その情報をシャーディング・メタデータに追加します。

手順全体を 1 つのメンテナンス時間枠 (つまり、アプリケーションがオフラインで、クラスタ上にユーザのアクティビティがない予定期間) で実行することも、ここで説明するように手順を 2 つのメンテナンス時間枠に分割することもできます。

詳細なステップを以下に示します。まだ十分に理解していない場合は、続行する前に "%SYSTEM.Cluster API を使用したクラスタの導入" のセクションを確認してください。"高可用性ガイド" の "ミラーの構成" の章で説明されているミラーの構成手順に関する知識も役に立ちますが、必須ではありません。この手順の各ステップには、必要に応じてこの章へのリンクが記載されています。

重要　ノード・レベルのミラーリングしていないクラスタを、この手順でミラー化したクラスタに変換するとネームスペース・レベルのクラスタになります。このクラスタの管理と変更は、%SYSTEM.Sharding API および管理ポータルのネームスペース・レベルのページでのみ可能です。

1. この手順を使用するには、クラスタのクラスタ・ネームスペースおよびマスタ・ネームスペースの名前がわかっている必要があります。これらの名前は、クラスタの導入時に指定されています。例えば、“管理ポータルを使用したクラスタの構成” の "データ・ノード 1 の構成" タスクの手順 4 で示されている手順では、クラスタ・ネームスペースとマスタ・ネームスペースの選択について説明しています。同様に、“%SYSTEM.Cluster API を使用したクラスタの構成” では、"ノード 1 の構成" の初期 API 呼び出しの説明に、クラスタ・ネームスペースとマスタ・ネームスペースの指定が含まれています。
2. ""%SYSTEM.Cluster API を使用したクラスタの導入"" の最初の 2 つのステップ "インフラストラクチャのプロビジョニングまたは特定" および "データ・ノードへの InterSystems IRIS の導入" の指示に従って、クラスタにバックアップ・フェイルオーバー・メンバとして追加するノードを準備します。想定されるバックアップのホストの特性と InterSystems IRIS の構成は、すべての点で既存のデータ・ノードと同じである必要があります ("高可用性ガイド" の "ミラー構成のガイドライン" を参照)。

   注釈　各フェイルオーバー・ペアに含まれる対象の最初のフェイルオーバー・メンバ (既存のデータ・ノード) および 2 番目のフェイルオーバー・メンバ (新しく追加したノード) をホスト名または IP アドレスで記録しておくと役に立ちます。

3. シャード・クラスタのメンテナンス時間枠を開始します。
4. 現在のデータ・ノードそれぞれで ISCAgent を開始し、ミラーを作成して最初のフェイルオーバー・メンバを構成します。
5. このクラスタをミラーリングされた構成に変換する、つまり、前のステップで作成したミラーをクラスタの構成およびメタデータに組み込むには、ノード 1 のインスタンスの InterSystems ターミナルを開き、以下のようにマスタ・ネームスペースで $SYSTEM.Sharding.AddDatabasesToMirrors() メソッドを呼び出します (″%SYSTEM.Sharding API″ を参照)。

   ```
   set status = $SYSTEM.Sharding.AddDatabasesToMirrors()
   ```

   注釈　これらの手順で詳述されている各 API 呼び出しの戻り値 (成功の場合は 1 など) を確認するには、以下を入力します。

   ```
   zw status
   ```

   状況によっては通知なしで呼び出しが失敗することがあるため、各呼び出しの後に **[ステータス]** を確認することをお勧めします。呼び出しが成功しなかった (**[ステータス]** が [1] 以外) 場合、以下を入力することにより、わかりやすいエラー・メッセージが表示されます。

   ```
   do $SYSTEM.Status.DisplayError(status)
   ```

   AddDatabasesToMirrors() 呼び出しは、以下を実行します。

   - ノード 1 でマスタ・データベースとシャード・データベースを追加し (“%SYSTEM.Cluster API を使用したクラスタの導入” の ″ノード 1 の初期化″ を参照)、他のデータ・ノードでシャード・データベースをそのそれぞれのミラーに追加します。
   - ノード間のすべての ECP 接続をミラー接続として再構成します。計算ノード (該当する場合) とその関連データ・ノード間の ECP 接続も対象です。
   - すべてのデータ・ノード上でリモート・データベースを再構成し、それに応じて関連するすべてのマッピングを調整します。
   - シャーディング・メタデータを更新し、再構成した接続、データベース、およびマッピングを反映させます。

   呼び出しが正常に完了すると、シャード・クラスタは完全に使用可能な状態になります (ただし、バックアップ・フェイルオーバー・メンバをまだ追加していないため、フェイルオーバーはまだ実行できません)。
6. データ・ノードの調整されたバックアップを実行します (つまり、すべてのノードが論理的な同じ時点でバックアップされているバックアップ)。具体的には、最初のフェイルオーバー・メンバ (既存のデータ・ノード) のそれぞれでシャード・データベースをバックアップし、ノード 1 でマスタ・データベースもバックアップします。バックアップの前に、以下のようにインスタンスの構成パラメータ・ファイル (CPF) を確認して、適切なデータベースを特定しておきます。

   - CPF の `[Map.clusternamespace]` セクションを探して (例えば、クラスタ・ネームスペースが **CLUSTERNAMESPACE** の場合、このセクションは `[Map.CLUSTERNAMESPACE]`)、**IRIS.SM.Shard** と **IS.*** グローバル接頭語のマッピング (このターゲットがシャード・データベース) を見つけることで、シャード・データベースを特定します。以下に示すように、他のグローバル接頭語もこのシャード・データベースにマッピングできます。シャード・データベースは **SHARDDB** として識別されます。

     ```
     [Map.CLUSTERNAMESPACE]
     Global_IRIS.SM.Shard=SHARDDB
     Global_IRIS.Shard.*=SHARDDB
     Global_IS.*=SHARDDB
     Package_IRIS.Federated=SHARDDB
     ```

- ノード 1 で、[Namespaces] セクションも見つけます。ここには、マスタ・ネームスペースの後に、その既定のグローバル・データベースとしてマスタ・データベースが表示されます。例えば、以下では、マスタ・データベース **MASTERDB** が、**MASTERNAMESPACE** の既定のグローバル・データベースとして表示されています。

```
[Namespaces]
%SYS=IRISSYS
CLUSTERNAMESPACE=SHARDDB
MASTERNAMESPACE=MASTERDB
USER=USER
```

7. オプションで、想定される 2 番目のフェイルオーバー・メンバと DR 非同期ミラー・メンバ (該当する場合) を以降のステップで準備する間、現在のメンテナンス時間枠を終了してアプリケーションのアクティビティを許可します。
8. 2 番目のフェイルオーバー・メンバまたは DR 非同期としてクラスタに追加する各ノードで、以下を実行します。
   a. ISCAgent を起動します。
   b. 目的の最初のフェイルオーバー・メンバ (前述の例では、**CLUSTERNAMESPACE**) 上のクラスタ・ネームスペースと同じ名前でネームスペースを作成し、その既定のグローバル・データベースとして、最初のフェイルオーバー・メンバ (例では、**SHARDDB**) 上のシャード・データベースと同じ名前でローカル・データベースを構成します。
   c. ノード 1 に追加する目的の 2 番目のフェイルオーバー・メンバまたは DR 非同期でも、目的の最初のフェイルオーバー・メンバ上のマスタ・ネームスペースと同じ名前でネームスペースを作成し、その既定のグローバル・データベースとして、マスタ・データベースと同じ名前でローカル・データベースを構成します。前述の例を使用すると、既定のグローバル・データベースとして **MASTERDB** という名前のデータベースを持つ **MASTERNAMESPACE** ネームスペースを作成することになります。
9. メンテナンス時間枠内ではない場合は、新しいメンテナンス時間枠を開始します。
10. 新しいノードそれぞれで、データが含まれるミラーリングされたデータベースを含む既存のミラーの 2 番目のフェイルオーバー・メンバまたは DR 非同期メンバとして、ミラーリングされていないインスタンスを追加するために必要なタスクを実行します。
    - 対象のミラーの 2 番目のフェイルオーバー・メンバとしてノードを構成するか、または (2 番目のフェイルオーバー・メンバの構成後に) ノードを DR 非同期メンバとして構成します。
    - 新しく構成したメンバ上で、最初のフェイルオーバー・メンバ上に作成したバックアップからシャード・データベースをリストアします。ノード 1 ミラー (2 番目のフェイルオーバーまたは DR 非同期) の新しく構成したメンバ上で、ノード 1 の最初のフェイルオーバー・メンバ上に作成したバックアップからマスタ・データベースもリストアします。
    - マスタ・データベースとクラスタ・データベースを有効化してキャッチアップします (オンライン・バックアップを使用してバックアップを作成した場合は必要ありません)。

    注釈　シャーディングによって自動的に、クラスタ・ネームスペース定義が更新され、必要なすべてのマッピング、ECP サーバ定義、およびリモート・データベース定義が作成され、マスタ・ネームスペース内のユーザ定義マッピングがシャードに伝播されます。したがって、このプロセスで手動で作成する必要があるマッピングは、マスタ・ネームスペース内のユーザ定義マッピングだけです。このマッピングは、ノード 1 の 2 番目のフェイルオーバー・メンバ上のマスタ・ネームスペース内にのみ作成する必要があります。ECP サーバ定義とリモート・データベース定義は手動でコピーする必要はありません。
11. いずれかのプライマリ (元のデータ・ノード) のインスタンスに対して InterSystems ターミナルを開き、以下のように、クラスタ・ネームスペース (またはノード 1 のマスタ・ネームスペース) で $SYSTEM.Sharding.VerifyShards() メソッドを呼び出します (″%SYSTEM.Sharding API″ を参照)。

```
set status = $SYSTEM.Sharding.VerifyShards()
```

この呼び出しにより、ミラーの 2 番目のフェイルオーバー・メンバについての必要な情報がシャーディング・メタデータに自動的に追加されます。

注釈　この呼び出しを実行する場合、元のすべてのクラスタ・ノードがそのミラーの現在のプライマリである必要があります。したがって、2 番目のフェイルオーバー・メンバの追加後にミラーがフェイルオーバーした場合は、このステップを実行する前に、元のフェイルオーバー・メンバへの計画的フェイルバックを準備する必要があります (計画的フェイルオーバーの手順は、"プライマリ・フェイルオーバー・メンバのメンテナンス" を参照してください。iris stop コマンドを使用して、その手順で参照されている適切なシャットダウンを行う方法は、"InterSystems IRIS インスタンスの制御" を参照してください)。

重要　上記の最後のステップが完了したら、メンテナンス時間枠を終了できます。ただし、クラスタをプロダクション環境に移行する前に、計画的フェイルオーバー (前の項目を参照) を実行して、各ミラーをテストすることを強くお勧めします。

3.5.2.5 変更をミラーリングするためのクラスタ・メタデータの更新

ここで説明されている API または管理ポータルによる手順以外の方法 (つまり、管理ポータルのミラーリングのページ、^MIRROR ルーチン、または SYS.MIRROR API) を使用して、ミラーリングされたクラスタ内の 1 つ以上のデータ・ノードのミラー構成を変更する場合は、クラスタのメタデータを更新する必要があります。そのためには、クラスタ・ネームスペース内で $SYSTEM.Sharding.VerifyShards() を呼び出すか ("%SYSTEM.Sharding API" を参照)、またはクラスタ内の現在のプライマリ・フェイルオーバー・メンバ上で管理ポータルの [Configure Node-Level] ページの **[シャードを検証]** ボタンを使用します ("管理ポータルを使用したミラーリングされたクラスタの構成" を参照)。例えば、計画的フェイルオーバーを実行する場合は、DR 非同期を追加するか、バックアップ・メンバを DR 非同期に降格させるか、または DR 非同期をフェイルオーバー・メンバに昇格させて、シャードによってメタデータが更新されて変更が反映されることを検証します。データ・ノード・ミラーに DR 非同期を含めることによって設定した災害復旧機能を維持・活用するうえで、クラスタ・メタデータを更新することは重要な要素です。詳細は、"ミラーリングされたシャード・クラスタの災害復旧" を参照してください。

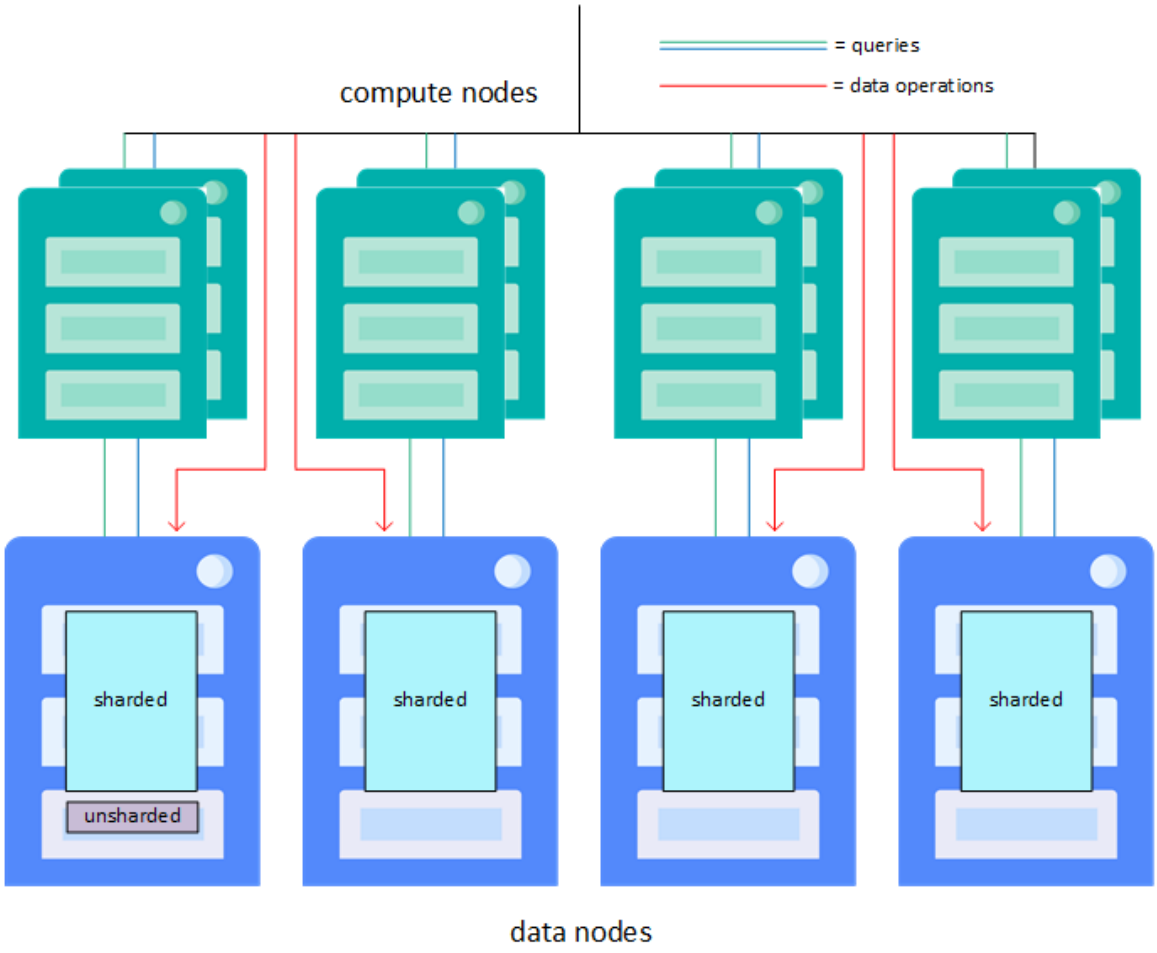
クラスタのシャードは、ミラーリング構成操作のたびに検証することも、一連の操作の完了後に一度だけ検証することもできますが、クラスタのオンライン中に操作を実行した場合は、フェイルオーバー・メンバを追加または削除する操作の実行後すぐにシャードを検証することをお勧めします。

3.5.3 作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入

常に大量のデータが取り込まれるような状況でも、クエリの遅延がきわめて小さいことが求められる高度な使用事例では、計算ノードを追加して、クエリを処理するための透過的なキャッシュ層を提供できます。各計算ノードは、それが関連付けられているデータ・ノード上のシャード・データと、必要に応じてシャード化されていないデータをキャッシュします。クラスタに計算ノードが含まれている場合は、データ・ノード上ではなく、その計算ノード上で読み取り専用クエリが自動的に並行して実行されます。データ・ノードではすべての書き込み操作 (挿入、更新、削除、および DDL 操作) が引き続き実行されます。この作業分担により、クエリとデータ取り込みの作業負荷が分離される一方で、並列処理や分散キャッシュの利点を維持し、これら両方のパフォーマンスを向上させます。1 つのデータ・ノードに複数の計算ノードを割り当てることで、クラスタのクエリ・スループットとパフォーマンスをさらに向上させることができます。

計算ノードがクラスタに追加されると、これらはデータ・ノード間でできる限り均等に自動分散されます。計算ノードを追加することによってパフォーマンスが大幅に向上するのは、計算ノードがデータ・ノードごとに少なくとも 1 つある場合のみです。クラスタの速度は最も低速なデータ・ノードと同じ速度にしかならないので、リソースの最も効率的な使用方法は、一般的に、各データ・ノードに同じ数の計算ノードを割り当てることです。計算ノードではクエリの実行のみがサポートされ、データは格納されないので、メモリと CPU を重視し、ストレージを最小限に抑えるなど、ニーズに合わせてハードウェア・プロファイルを調整できます。

図 3-6: 計算ノードを含むシャード・クラスタ



計算ノード、および計算ノードを使用したアプリケーションのクラスタへの接続の負荷分散の詳細は、"計算ノードの計画"を参照してください。

"シャード・クラスタの導入" で説明されているいずれかの導入方法を使用して、シャード・クラスタに計算ノードを追加できます。そこで説明されている自動導入方法にはすべて、計算ノードがオプションとして含まれます。このセクションでは、計算ノードを手動で導入する追加手順を説明します。まず、上記のセクションで説明されているステップを使用して、シャード・クラスタを手動で導入および構成します。その後、ここで説明するステップを使用して導入を完了します。

1. データ・ノードの計画
2. データベース・キャッシュとデータベースのサイズの見積もり
3. インフラストラクチャのプロビジョニングまたは特定

   注釈　計画した計算ノードのホストを、計画したデータ・ノードのホストと共に含めます。例えば、8 個のデータ・ノードと 8 個の計算ノードが必要な計画の場合、クラスタには 16 個のホストが必要です。シャード・クラスタ内のすべてのノードの仕様とリソースが同じであるか、少なくともほぼ同等である必要があります。ただし、計算ノードのストレージは例外です。計算ノードはシャード・クラスタ・ロールではストレージを使用しません。

4. データ・ノード・ホストへの InterSystems IRIS の導入
5. 管理ポータルまたは %SYSTEM.Cluster API を使用したデータ・ノードの構成
6. 管理ポータルまたは %SYSTEM.Cluster API を使用した計算ノードの追加 (以下の各セクションを参照)

3.5.3.1 管理ポータルを使用した計算ノードの構成または導入

クラスタに計算ノードを追加すると、データ・ノード間で計算ノードを自動的に分散するため、計算ノードは、前に関連付けられていた計算ノード数が最小だったデータ・ノードに割り当てられます。この手順は、クラスタをミラーリングするかどうかに関係なく同じです。

ブラウザで管理ポータルを開く方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎：IDE の接続" の "InterSystems IRIS 接続情報" のセクションで、キットからインストールしたインスタンスの手順を参照してください。

ネットワーク・システム上のインスタンスを計算ノードとしてクラスタに追加するには、以下の手順を使用します。

1. インスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→** [Sharding]**→[シャード有効]** の順に選択し、表示されるダイアログで [OK] をクリックします(既定値はほぼすべてのクラスタに適しているため、**[ECP 接続の最大数]** 設定の値を変更する必要はありません)。
2. インスタンスを再起動します(管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動した後に再読み込みするだけでかまいません)。
3. [Configure Node-Level] ページ (**[システム管理]→[構成]→[システム構成]→**[Sharding]**→**[Configure Node-Level]) に移動して、**[構成]** ボタンをクリックします。
4. [Configure Node-Level Cluster] ダイアログで [Add this instance to an existing sharded cluster] を選択し、表示されるプロンプトに次のように応答します。
   - クラスタ URL を入力します。この URL は、既にクラスタに属しているインスタンスの [Configure Node-Level] ページの **[シャード]** タブに、すべてのノードに対して表示されるアドレスです。"管理ポータルを使用したクラスタの導入" の "残りのデータ・ノードの構成" のステップを参照してください。

     注釈　クラスタがミラーリングされている場合は、プライマリ・データ・ノードまたは計算ノードのアドレスを入力します。バックアップ・データ・ノードのアドレスではありません。

   - **[ロール]** プロンプトで **[計算]** を選択して、インスタンスをデータ・ノードとして構成します。
   - 場合によっては、InterSystems IRIS に認識されているホスト名が適切なアドレスに解決されないことや、ホスト名が利用できないことがあります。このような理由または他の理由で、代わりに IP アドレスを使用して他のクラスタ・ノードをこのノードと通信させたい場合は、**ホスト名のオーバーライド**のプロンプトで IP アドレスを入力します。
   - [Mirrored cluster] チェックボックスは利用できません。計算ノードの構成はミラーリングに関係なく同じであるためです (ただし、前述のようにクラスタ URL が指定されている場合を除きます)。
5. [OK] をクリックして [Configure Node-Level] ページに戻ります。2 つのタブ **[シャード]** と **[シャードテーブル]** が表示されています。これまでに構成したデータ・ノードと計算ノードが **[シャード]** にノード 1 から表示され、各計算ノードがどのデータ・ノードに割り当てられているかを示します。

   [シャードを検証] をクリックして、計算ノードが正しく構成されていて、他のノードと通信できることを検証します。

   注釈　構成する計算ノードが多数ある場合は、**[詳細設定]** ボタンをクリックし、[詳細設定] ダイアログで **[割り当て時に自動的にシャードを検証]** を選択することにより、検証操作を自動化できます(シャード・クラスタを導入する際は、このダイアログのその他の設定は既定値のままにします)。

3.5.3.2 %SYSTEM.Cluster API を使用した計算ノードの導入

ネットワーク・システム上のインスタンスをクラスタに計算ノードとして追加するには、そのインスタンスの InterSystems ターミナルを開き、既存のクラスタ・ノードのホスト名とその InterSystems IRIS インスタンスのスーパーサーバ・ポートを指定して $SYSTEM.Cluster.AttachAsComputeNode() メソッドを呼び出します。以下に例を示します。

```
set status = $SYSTEM.Cluster.AttachAsComputeNode("IRIS://datanode2:1972")
```

注釈　これらの手順で詳述されている各 API 呼び出しの戻り値 (成功の場合は 1 など) を確認するには、以下を入力します。

```
zw status
```

呼び出しが成功しなかった場合、以下を入力することにより、わかりやすいエラー・メッセージが表示されます。

```
do $SYSTEM.Status.DisplayError(status)
```

テンプレート・ノードの構成時にこの IP アドレスを指定した場合 (“%SYSTEM.Cluster API を使用したクラスタの導入” の ″ノード 1 の構成″ を参照) は、ホスト名ではなく、その IP アドレスを使用します。

```
set status = $SYSTEM.Cluster.AttachAsComputeNode("IRIS://100.00.0.01:1972")
```

他のノードに、IP アドレスを使用してこのノードと通信させる場合は、2 番目の引数として IP アドレスを指定します。

注釈　もう 1 つのノード (この手順で必要なノード) から見ると、コンテナ化された InterSystems IRIS インスタンスのスーパーサーバ・ポートは、そのコンテナが作成されたときにスーパーサーバ・ポートが公開されたホスト・ポートによって異なります。この詳細および例は、″コンテナ内でのインターシステムズ製品の実行″ の ″永続的な %SYS を使用した InterSystems IRIS コンテナの実行″ と ″InterSystems IRIS コンテナの実行 : Docker Compose の例″、および Docker ドキュメントの ″Container networking″ を参照してください。

そのホスト上にしかないコンテナ化されていない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。インスタンスのスーパーサーバ・ポート番号を表示または設定するには、そのインスタンスの管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します (インスタンスの管理ポータルを開いてスーパーサーバ・ポートを確認する方法の詳細は、コンテナに導入されたインスタンスの手順、または ″InterSystems IRIS の基礎 : IDE の接続″ の ″InterSystems IRIS 接続情報″ で、キットからインストールしたインスタンスの手順を参照してください)。

最初の引数で指定したクラスタ・ノードがデータ・ノードの場合、これがテンプレートとして使用されます。計算ノードの場合は、これが割り当てられているデータ・ノードがテンプレートとして使用されます。AttachAsComputeNode() 呼び出しは、以下を実行します。

- ECP およびシャーディング・サービスを有効にします。
- データ・ノード間で計算ノードを自動的に分散するため、新しい計算ノードを、前に関連付けられた計算ノード数が最小だったデータ・ノードに関連付けます。
- クラスタ・ネームスペースを作成し、“%SYSTEM.Cluster API を使用したクラスタの構成” の ″ノード 1 の構成″ で説明したとおり、それをテンプレート・ノード (最初の引数で指定) の設定に合わせて構成し、必要なすべてのマッピングを作成します。
- すべての SQL 構成オプションを、テンプレート・ノードに合わせて設定します。

クラスタ・ネームスペースと同じ名前のネームスペースが新しい計算ノードに既に存在する場合は、そのネームスペースがクラスタ・ネームスペースとして使用され、マッピングのみが複製されます。

他のクラスタ・ノードに、ホスト名ではなく IP アドレスを使用してこのノードと通信させる場合は、2 番目の引数として IP アドレスを指定します。

AttachAsComputeNode() 呼び出しは、InterSystems IRIS インスタンスが既にシャード・クラスタのノードである場合、エラーを返します。

すべての計算ノードを構成したら、$SYSTEM.Cluster.ListNodes() メソッドを呼び出してこれらをリストできます。以下に例を示します。

```
set status = $system.Cluster.ListNodes()
NodeId  NodeType    DataNodeId    Host          Port
1       Data                      datanode1     1972
2       Data                      datanode2     1972
3       Data                      datanode3     1972
1001    Compute     1             computenode1  1972
1002    Compute     2             computenode2  1972
1003    Compute     3             computenode3  1972
```

計算ノードが導入されると、このリストに各計算ノードが割り当てられるデータ・ノードのノード ID が示されます。$SYSTEM.Cluster.GetMetadata() を使用して、クラスタおよびマスタ・ネームスペースの名前、それらの既定のグローバル・データベース、および呼び出しを発行するノードの設定を含む、クラスタのメタデータを取得することもできます。

3.5.4 1 つのホストへの複数のデータ・ノードのインストール

指定数のシステムでデータ・ノードをホストしている場合、%SYSTEM.Shardng API を使用してホストごとに複数のデータ・ノード・インスタンスを構成すると、データ取り込みのスループットが大幅に向上します (%SYSTEM.Cluster API や管理ポータルでは、このような構成は不可能です)。したがって、最小限のコストで最大のデータ取り込みのスループットを達成することが重要である場合、ホストあたり 2 つまたは 3 つのデータ・ノード・インスタンスを構成することにより、これを実現できます。この方法で得られる効果は、サーバ・タイプ、サーバ・リソース、および全体の作業負荷によって異なります。システムの総数を増やすことで (ホスト・システムのメモリを複数のデータベース・キャッシュ間で分割することなく) 同様の、またはそれ以上のスループットの向上を実現できますが、インスタンスの追加は、システムの追加と比べてコストがかかりません。

3.6 InterSystems IRIS シャーディング・リファレンス

このセクションでは、シャード構成の計画、導入、および使用に関する追加情報を紹介します。内容は以下のとおりです。

- InterSystems IRIS シャード・クラスタの計画
- シャード・クラスタの調整されたバックアップとリストア
- ミラーリングされたシャード・クラスタの災害復旧
- シャード・クラスタの複製
- シャーディング API
- ネームスペース・レベル・アーキテクチャの導入
- 予約名

3.6.1 InterSystems IRIS シャード・クラスタの計画

このセクションでは、基本的なシャード・クラスタの計画と、計算ノードの追加 (該当する場合) に関する一次ガイドラインを紹介します。完全な設計および計画プロセスに関する詳細な説明を目的としたものではありません。以下のタスクについて説明します。

- シャーディングと垂直方向の拡張の併用
- データ・ノードの基本的なクラスタの計画
- 計算ノードの計画

3.6.1.1 シャーディングと垂直方向の拡張の併用

シャーディングについて計画する際には、一般に、システムごとのリソースと使用するシステム数との間のトレードオフを考慮する必要があります。極端に言えば、2 つの主なアプローチを以下のように説明できます。

- 垂直方向に拡張して、各システムおよびインスタンスの性能をできるだけ高めてから、高性能なノードを追加して水平方向に拡張します。
- 高度な構成の高性能な 1 つのシステムに代わるコスト効率の高い手段として、手ごろな価格であるが、性能が劣るシステムを複数使用して水平方向に拡張します。

実際には、ほとんどの場合、これらのアプローチを組み合わせると最も効果を発揮します。他の水平方向の拡張アプローチとは異なり、InterSystems IRIS のシャーディングは、InterSystems IRIS の優れた垂直方向の拡張性と容易に組み合わせることができます。多くの場合、4 ～ 16 台のデータ・ノードを含む、処理能力がかなり高いシステムでホストされているクラスタで最も大きな成果が得られます。

3.6.1.2 データ・ノードの基本的なクラスタの計画

下記のガイドラインを使用するには、クラスタに格納するデータの量に関連するいくつかの変数を見積もる必要があります。

1. まず、クラスタに格納するデータを検討して、以下について見積もりを行います。
   a. クラスタに格納するシャード・テーブルすべて (インデックスを含む) の合計サイズ
   b. クラスタに格納するシャード化されていないテーブルのうち、シャード・テーブルと頻繁に結合されるもの (インデックスを含む) の合計サイズ
   c. クラスタに格納するシャード化されていないすべてのテーブル (インデックスを含む) の合計サイズ。(前の見積もりはこの見積もりのサブセットです。)

2. 定期的にクエリが実行されるデータの割合に基づいて、これらの合計を、予測される作業セットに変換します。

   作業セットの見積もりは、複雑な問題である場合があります。既存のデータベース・キャッシュの過去の使用統計から、これらの作業セットに関する有用な情報を取得できることがあります。それに加えて、またはその代わりに、テーブルを 3 つのカテゴリに分け、以下の作業を行うことにより、それぞれについて大まかな作業セットを特定します。

   - テーブルに対して頻繁に実行される重要な SELECT 文について、WHERE 節を確認します。それらは通常、テーブルおよび列の統計に基づいてサイズを見積もることができる可能性があるデータのサブセットを調べますか。個々の SELECT 文によって取得されるサブセットは相互に重複していますか、それとも相加的ですか。
   - 重要な INSERT 文のサイズと頻度を確認します。これらを作業セットに変換するのはさらに難しい場合がありますが、簡単なアプローチとして、1 時間あたりの平均取り込み速度を MB 単位で見積もり (1 秒あたりのレコード数 * レコードの平均サイズ * 3600)、それをテーブルの作業セットに加算します。
   - 返される結果を具体的に見積もることができる可能性がある他の高頻度のクエリを検討します。
   - シャード化されていないテーブルとシャード・テーブルを結合するクエリは作業セット NonshardSizeJoinedWS に反映され、シャード化されていない同じデータ・テーブルに対するクエリのうち、シャード化されていないテーブルをシャード・テーブルに結合しないものは作業セット NonshardSizeTotalWS に反映される点に留意してください。シャード化されていない同じデータが両方のタイプのクエリによって返されることがあるため、両方の作業セットに反映されます。

   その後、これらの見積もりを合計して各テーブルの作業セットについて 1 つの見積もりを作成し、それらの見積もりを加算することで全体的な作業セットを大まかに計算できます。これらの全体的な見積もりはかなり大まかである可能性があり、プロダクション環境では調整が必要になることがあります。50% の安全係数を各見積もりに加算し、最終的な合計データ・サイズと作業セットを以下の変数として記録します。

テーブル 3-1: クラスタの計画に関する変数

変数	値
ShardSize、ShardSizeWS	シャード・テーブルの合計サイズと作業セット（安全係数を含む）
NonshardSizeJoined、NonshardSizeJoinedWS	シャード・テーブルに頻繁に結合されるシャード化されていないテーブルの合計サイズと作業セット（安全係数を含む）
NonshardSizeTotal、NonshardSizeTotalWS	シャード化されていないテーブルの合計サイズと作業セット（安全係数を含む）
NodeCount	データ・ノード・インスタンスの数

下記のテーブルのガイドラインを確認する際には、以下の点に留意してください。

- 一般的に言えば、他の条件がすべて同じであれば、シャードが多い方が並列処理が増えるため、パフォーマンスが向上します。ただし、通常はデータ・ノードが 16 台ほどになるとオーバーヘッドが発生するため、パフォーマンスは低下します。
- 紹介しているガイドラインは、最小要件ではなく、理想的、または最も効果的な構成を表しています。

  例えば、"シャーディングの効果の評価" に記載しているように、シャーディングによってパフォーマンスが向上するのは、1 つには、シャード化されていない単一インスタンスですべてのデータがキャッシュされるのではなく、複数のシステム間でデータがキャッシュされることによります。定期的に使用されるデータが大きすぎて、シャード化されていないインスタンスのデータベース・キャッシュに収まらない場合、効果が最も大きくなります。ガイドラインに示しているように、最適なパフォーマンスを得るには、クラスタ内の各データ・ノード・インスタンスのデータベース・キャッシュが、シャード・データ作業セットと頻繁に結合されるシャード化されていないデータ作業セットの合計サイズ以上になるようにします。合計キャッシュ・サイズが小さくなると、パフォーマンスが低下します（他の条件がすべて同じである場合）。ただし、すべてのデータ・ノード・キャッシュの合計が、シャード化されていない指定の単一インスタンスのキャッシュ・サイズ以上である限り、シャード・クラスタは、シャード化されていないインスタンスよりも優れたパフォーマンスを発揮します。したがって、例えば、推奨値と等しいデータベース・キャッシュ・メモリをデータ・ノードで割り当てることができない場合は、できる限り近づけてください。さらに、初期見積もりは、実践では調整が必要になることがあります。
- データベース・キャッシュとは、各インスタンスに対して行う必要があるデータベース・キャッシュ（グローバル・バッファ・プール）のメモリ割り当てを指します。インスタンスのデータベース・キャッシュは、以下のように割り当てることができます。
  - いずれかの自動導入方法を使用する場合、導入の一部として既定の globals 設定をオーバーライドできます。
  - %SYSTEM.Cluster API または管理ポータルを使用して手動で導入する場合は、"システム管理ガイド" の "メモリと開始設定" で説明されている手動手順を使用できます。

  InterSystems IRIS インスタンスのルーチン・キャッシュとデータベース・キャッシュおよび共有メモリ・ヒープにメモリを割り当てる際のガイドラインは、"メモリ要件の見積もり" を参照してください。
- 既定のグローバル・データベースは、関連するデータベースのターゲット・サイズを示します。これは、予想される最大サイズに、予想を超える増大を見越したマージンを加えたものです。データベースをホストするファイル・システムは、この合計に対応できるサイズでなければなりません。また、安全マージンも確保する必要があります。InterSystems IRIS データベースのサイズと拡張、InterSystems IRIS データベースに対する空き容量の管理、および手動でインスタンスを構成する際のデータベース・サイズやその他特性の指定手順についての一般情報は、"システム管理ガイド" の “InterSystems IRIS の構成” の章にある "データベースの構成" および “InterSystems IRIS の管理” の章にある "ローカル・データベースの管理" を参照してください。

IKO を使用して導入する場合、インスタンスのデータ用のストレージ・ボリューム・サイズを指定できます。これはマスタ・ネームスペースとクラスタ・ネームスペースの既定のグローバル・データベースが導入の一部として配置される場所です。これは、既定のグローバル・データベースのターゲット・サイズに対応できる十分な大きさである必要があります。

重要　手動で導入する場合、すべてのインスタンスでデータベース・ディレクトリとジャーナル・ディレクトリが個別のストレージ・デバイスに配置されるようにします。これは、大量のデータ取り込みがクエリの実行と同時に行われる場合に特に重要です。ファイル・システムの構成およびジャーナル・ストレージなどのストレージの構成のガイドラインについては、"システム・リソースの計画と管理" の "ストレージの計画" と "ファイル・システムの分離"、および "データ整合性ガイド" の "ジャーナリングの最善の使用方法" を参照してください。

- データ・ノードの数 (NodeCount) と各データ・ノードのデータベース・キャッシュ・サイズは、どちらも変数です。シャードの数と各データ・ノードのデータベース・キャッシュ・サイズを変えることにより、データ・ノードのデータベース・キャッシュ・サイズの合計と合計作業セットの見積もりの間に必要な関係を作成できます。この選択は、システム・コストとメモリ・コストの間のトレードオフなどの要因に基づいて行うことができます。メモリ・リソースが少ない多数のシステムが使用可能な場合は、少量のメモリをデータベース・キャッシュに割り当てることができます。システムごとのメモリが多い場合は、少数のサーバを構成できます。一般的に言えば、他の条件がすべて同じであれば、シャードが多い方が並列処理が増えるため、パフォーマンスが向上します。ただし、ある時点になると、パフォーマンスは低下します (原因の一部は、増加したシャーディング・マネージャのオーバーヘッド)。通常、最も効果的な構成は、シャードが 4 ～ 16 の範囲である場合です。つまり、他の要因を除くと、例えば、8 台のデータ・ノードがあり、それぞれのメモリが M である場合、64 台のシャードがあり、それぞれのメモリが M/8 である場合よりも優れたパフォーマンスを発揮する可能性があります。
- クラスタにデータがロードされた後にデータ・ノードを追加する必要がある場合は、シャード・データを新しいサーバにわたって自動的に再分散できることに留意してください。これにより、パフォーマンスが最適化されます。詳細は、"データ・ノードの追加とデータの再分散" を参照してください。一方、シャード・データがあるデータ・ノードは削除できず、サーバのシャード・データは自動的に別のデータ・ノードに再分散できません。したがって、プロダクションのクラスタへのデータ・ノードの追加は、データ・ノードの数の削減 (すべてのシャード・テーブルを削除してからデータ・ノードを削除し、データを再ロードする) に比べ、かなり少ない労力ですみます。
- クエリの並列処理の速度は、最も低速なデータ・ノードと同じ速度にしかなりません。したがって、シャード・クラスタ内のすべてのデータ・ノードの仕様とリソースが同じであるか、少なくともほぼ同等であるようにすることをお勧めします。さらに、クラスタ内のすべての InterSystems IRIS インスタンスの構成が一貫している必要があります。確実に正しい SQL クエリ結果が返されるようにするには、インスタンス・レベルで構成される照合などのデータベース設定やそれらの SQL 設定 (既定の日付形式など) がすべてのノードで同じでなければなりません。標準化された手順や自動導入方法を使用すると、この一貫性を簡単に確保できます。

以下のテーブルの推奨事項は、前述の合計データ・サイズと作業セット・サイズの見積もり手順を、各変数について計算結果に 50% の安全係数を加算することを含め、実行済みであることを前提としています。

テーブル 3-2: クラスタの計画に関するガイドライン

対象	必要な最小サイズ	注
データ・ノード上のデータベース・キャッシュ	(ShardSizeWS / NodeCount) + NonshardSizeJoinedWS	この推奨は、アプリケーションで 100% のメモリ内キャッシュを必要としていることを前提としています。代わりにソリッドステート・ドライブなどの高速ストレージから読み取ることのできる範囲によっては、キャッシュのサイズを削減できます。
各データ・ノードのクラスタ・ネームスペースの既定グローバル・データベース	ShardSize / NodeCount に、予想される増大に対応するためのスペースを加えたもの	データ取り込みのパフォーマンスが重要な考慮事項である場合は、データベースの初期サイズを予想される最大サイズと等しくなるように構成し、自動的なデータベースの拡張によるパフォーマンスの影響を回避することを検討してください。ただし、クラウド環境で実行している場合は、使用していないストレージの支払いによるコスト面での影響も検討する必要があります。
ノード 1 上のマスタ・ネームスペースの既定のグローバル・データベース (″ネームスペースの構成″ を参照)	NonshardSizeTotal および、可能であれば、予想される増大に対応するためのスペース	シャード化されていないデータは、シャード・データに比べ、時間が経ってもあまり増大しない傾向がありますが、もちろんこれは、アプリケーションによります。
各データ・ノード上の IRISTEMP データベース (マスタ・ネームスペースおよびクラスタ・ネームスペース用の一時ストレージ・データベース)	特定のガイドラインはありません。理想的な初期サイズはデータ・セット、作業負荷、およびクエリ構文により異なりますが、おそらく 100 GB を超え、これよりはるかに大きい場合もあります。	このデータベースは必ず、大幅な拡張に対応できる大きな領域がある、可能な限り最速のストレージに配置してください。T
CPU	具体的な推奨値はありません。	InterSystems IRIS サーバはすべて、シャーディングが含まれるかどうかに関係なく、CPU を増やすことでパフォーマンスが向上します。CPU、メモリ、およびストレージ・リソースの垂直方向の拡張は、常に、シャーディングと組み合わせて使用することでさらに大きな効果をもたらしますが、特に必要ではなく、一般的なコストとパフォーマンスのトレードオフによって制御されます。

重要

シャード・クラスタ内のすべての InterSystems IRIS インスタンスは、同じバージョンである必要があり、シャーディング・ライセンスを有する必要があります。

シャード・クラスタ内のすべてのデータ・ノードの仕様とリソースが同じであるか、少なくともほぼ同等である必要があります。クエリの並列処理の速度は、最も低速なデータ・ノードと同じ速度にしかなりません。さらに、クラスタ内のすべての InterSystems IRIS インスタンスの構成が一貫している必要があります。確実に正しい SQL クエリ結果が返されるようにするには、インスタンス・レベルで構成される照合などのデータベース設定やそれらの SQL 設定 (既定の日付形式など) がすべてのノードで同じでなければなりません。標準化された手順や自動導入方法を使用すると、この一貫性を簡単に確保できます。

アプリケーションは任意のデータ・ノードのクラスタ・ネームスペースに接続して、全データセットをローカルのものであるかのように利用できるので、一般的なベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード間でアプリケーション接続を負荷分散することをお勧めします。IKO は、一般的なシナリオの場合、必要に応じてデータ・ノードのロード・バランサを自動的にプロビジョニングし、構成します。他の手段でシャード・クラスタを導入する場合は負荷分散メカニズムが必要となります。複数のデータ・ノードにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明は、"Web ゲートウェイ・ガイド" の "負荷分散、フェイルオーバー、ミラー構成" を参照してください。

クラスタのパフォーマンスを最大限に発揮させるためのベスト・プラクティスは、すべてのデータ・ノード間に低遅延のネットワーク接続を構成することです。例えば、すべてのデータ・ノードを同じデータ・センターまたはアベイラビリティ・ゾーン内の同じサブネットに配置します。

3.6.1.3 計算ノードの計画

"InterSystems IRIS のシャーディングの概要" に記載されているように、計算ノードがデータ・ノードに格納されているデータをキャッシュし、自動的に読み取り専用クエリを処理する一方、すべての書き込み操作 (挿入、更新、削除、および DDL 操作) はデータ・ノードで実行されます。計算ノードをクラスタに追加することで効果が得られる可能性が最も高いシナリオは、以下のとおりです。

- 大量のデータ取り込みが大量のクエリの実行と同時に行われる場合、データ・ノードあたり 1 つの計算ノードにすると、データ取り込みの作業負荷 (データ・ノード) からクエリの作業負荷 (計算ノード) が分離され、パフォーマンスが向上します。
- 多数のユーザによる大量のクエリがパフォーマンス低下の重大な要因である場合、データ・ノードあたり複数の計算ノードとすることで、基盤となる各データ・ノード上のデータに対して複数の同時シャード・クエリを実行できるようになり、全体的なクエリのスループット (および結果としてパフォーマンス) が向上します(一度に 1 つずつ、個別のシャード・クエリを実行する際のパフォーマンスは、計算ノードが複数になっても向上しません。多数のユーザによるクエリの作業負荷が関与しない限り、これは有益ではないためです)。計算ノードが複数あると、1 つの計算ノードに障害が発生した場合に、そのデータ・ノードに割り当てられた残りの計算ノードでクエリを処理できるため、作業負荷の分離を維持することもできます。

計算ノードを計画する際は、以下の要素について検討します。

- 計算ノードの導入を検討する場合、一般的には、基本的なシャード・クラスタの動作を評価してから、クラスタにとって計算ノードの追加が有益であるかどうかを決定することをお勧めします。計算ノードは、"クラスタの自動導入方法" で説明されている自動導入方法の 1 つを使用するか、%SYSTEM.Cluster API を使用して、既存のクラスタに簡単に追加できます。計算ノードの追加の詳細は、"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" を参照してください。
- 最適なパフォーマンスを得るには、クラスタの計算ノードをデータ・ノードと同じ場所 (つまり、同じデータ・センターまたはアベイラビリティ・ゾーン内) に配置して、ネットワーク遅延を最小化する必要があります。
- 計算ノードがクラスタに追加されると、これらはデータ・ノード間でできる限り均等に自動分散されます。計算ノードを追加することによってパフォーマンスが大幅に向上するのは、データ・ノードごとに計算ノードが少なくとも 1 つある場合のみであることに注意してください

- 各データ・ノードに同じ数の計算ノードを割り当てることをお勧めします。したがって、例えば 8 つのデータ・ノードを計画している場合、計算ノード数の推奨選択肢には、0、8、16 などが含まれます。
- 計算ノードではクエリの実行のみがサポートされ、データは格納されないので、メモリと CPU を重視し、ストレージを最小限に抑えるなど、ニーズに合わせてハードウェア・プロファイルを調整できます。シャード・クラスタ内のすべての計算ノードの仕様とリソースは、ほぼ同等である必要があります。
- 計算ノードに対し、データ・ノードのデータベース・キャッシュ・サイズの推奨事項 (″データ・ノードの基本的なクラスタの計画″ を参照) に従います。各計算ノードが、割り当てられているデータ・ノードと同じサイズのデータベース・キャッシュを持つのが理想的です。

データ・ノードと計算ノードの区別は、アプリケーションに対して完全に透過的です。アプリケーションは任意のノードのクラスタ・ネームスペースに接続して、全データセットをローカルのものであるかのように利用できます。したがって、アプリケーションの接続は、クラスタ内のすべてのデータ・ノードおよび計算ノードで負荷分散でき、ほとんどのアプリケーション・シナリオで、これが最も効果的なアプローチです。特定のシナリオで実際に何が最適であるかは、クエリの処理とデータの取り込みのどちらを最適化したいのかにより異なります。シャード・クエリが最も重要である場合は、計算ノードのリソースについて、アプリケーションがシャードローカル・クエリと競合しないように、データ・ノード間で負荷分散することにより、シャード・クエリを優先させることができます。並列ロードを使用した高速取り込みが最も重要である場合は、計算ノード間で負荷分散することで、データ・ノードでのアプリケーション・アクティビティを回避できます。クエリとデータ取り込みが同じように重要である場合、またはそれらを組み合わせた状態が予測できない場合は、すべてのノードで負荷分散します。

IKO では、データ・ノードまたは計算ノードの定義に自動的にロード・バランサを追加できます。また、独自の負荷分散配置を作成することもできます。複数のデータ・ノードにわたってアプリケーション接続を分散する Web サーバ層の負荷分散に関する重要な説明は、″Web ゲートウェイ・ガイド″ の ″負荷分散、フェイルオーバー、ミラー構成″ を参照してください。

3.6.2 シャード・クラスタの調整されたバックアップとリストア

InterSystems IRIS シャード・クラスタの場合のように、複数のシステムにわたってデータを分散する場合、バックアップおよびリストアの手順で複雑さが増す可能性があります。シャード・クラスタ間のデータの厳格な整合性が必要な場合、個々のノードを個別にバックアップしてリストアすることでは不十分です。バックアップすべてが、論理的な同じ時点で作成されているとは限らないからです。これにより、障害の後にクラスタ全体をリストアする際に、確実に順序を保持し、それによってリストアしたデータベースの論理整合性を確保することは不可能になります。

例えば、データ・ノード S1 上のデータの更新 A が、データ・ノード S2 上のデータの更新 B より前に実行されたとします。バックアップからのクラスタのリストアの後に、論理整合性では、更新 B が認識可能であれば、更新 A も認識可能である必要があります。ところが、S1 と S2 のバックアップが個別に実行されると、S2 のバックアップが B のコミット後に行われても、S1 のバックアップが A のコミット後に行われたことを保証するのは不可能です。したがって、バックアップを個別にリストアすると、S1 と S2 が互いに不整合になる可能性があります。

一方、使用した手順でバックアップまたはリストアを調整して、すべてのシステムが論理的な同じ時点 (ここでは、更新 B の後) にリストアされることを確実にすると、順序が保持されて、その順序に依存する論理整合性が確保されます。これは、調整されたバックアップおよびリストアの手順の目標です。

ここで説明したシャード・クラスタのリストアのいずれかの手順を使用しなければならなくなる可能性を大幅に小さくするために、″ミラーによる高可用性″ で説明しているように、ミラーリングされたデータ・サーバを使用してシャード・クラスタを導入できます。クラスタがミラーリングされていない場合でも、ほとんどのデータ・エラー (例えば、データ破損やデータの誤った削除) は、エラーが発生したデータ・ノードを最新のバックアップからリストアし、そのジャーナル・ファイルを使用して現在の論理的な時点に回復することで修正できます。ここで説明した手順は、クラスタ全体のリストアが必要なきわめてまれな状況で使用するものです。

このセクションでは、以下のトピックについて説明します。

- シャード・クラスタの調整されたバックアップとリストアのアプローチ
- API 呼び出しの調整されたバックアップとリストア

- 調整されたバックアップとリストアの手順

3.6.2.1 シャード・クラスタの調整されたバックアップとリストアのアプローチ

シャード・クラスタの調整されたバックアップとリストアには、常にクラスタ内のすべてのデータ・ノードが関係します。InterSystems IRIS バックアップ API には、クラスタのデータ・ノードの調整されたバックアップとリストアを行う 3 種類のアプローチをサポートする Backup.ShardedCluster クラスが含まれます。

すべてのアプローチの目標が、すべてのデータ・サーバを同じ論理的な時点にリストアすることであり、それを行う方法が異なるだけです。1 つのアプローチでは、論理的な時点を共有するのはバックアップ自体ですが、それ以外のアプローチでは、InterSystems IRIS データベース・ジャーナリングが共通の論理的な時点を提供します。これはジャーナル・チェックポイントと呼ばれ、ここにデータベースがリストアされます。以下のようなアプローチがあります。

- 調整されたバックアップ
- 調整されていないバックアップの後の、調整されたジャーナル・チェックポイント
- 調整されていないバックアップに含まれる調整されたジャーナル・チェックポイント

これらのアプローチが機能するしくみを理解するには、InterSystems IRIS データ整合性とクラッシュ回復の基本を理解することが重要です。これについては、"データ整合性ガイド" の "データ整合性の概要" の章で説明されています。データベース・ジャーナリングは、データ整合性および回復の重要な機能であり、特にこのトピックでは大きな影響があります。ジャーナリングでは、インスタンスのデータベースに加えられたすべての更新がジャーナル・ファイルに記録されます。これによって、バックアップが作成された時点から障害が発生した時点 (または別の選択した時点) までの間に加えられた更新を回復できるようになります。これは、バックアップからのリストアの後にジャーナル・ファイルから更新をリストアすることで実行できます。ジャーナル・ファイルは、トランザクション整合性の確保にも使用され、障害によって開いたままになっているトランザクションをロールバックすることで実現されます。ジャーナリングの詳細は、"データ整合性ガイド" の "ジャーナリング" の章を参照してください。

調整されたバックアップとリストアのアプローチを選択する際の考慮事項は、以下のとおりです。

- アクティビティがバックアップ手順によって中断される度合い
- 十分な復元可能性を保証するために必要な、バックアップ手順の実行頻度
- 必要なリストア手順の複雑さ

これらの問題の詳細は、このセクションで後述します。

3.6.2.2 API 呼び出しの調整されたバックアップとリストア

Backup.ShardedCluster クラス内のメソッドは、任意のデータ・ノードで呼び出すことができます。メソッドのすべてが ShardMasterNamespace 引数を取ります。これは、クラスタ内のすべてのノードからアクセスできる、データ・ノード 1 上のマスタ・ネームスペースの名前 (既定では **IRISDM**) です。

使用できるメソッドは以下のとおりです。

- Backup.ShardedCluster.Quiesce()

  シャード・クラスタのすべてのデータ・ノード上のすべてのアクティビティをブロックし、保留中の書き込みすべてがディスクにフラッシュされるまで待機します。Quiesce() の下に作成される、クラスタのデータ・ノードのバックアップは、論理的な時点を表します。

- Backup.ShardedCluster.Resume()

  Quiesce() によって一時停止されているデータ・ノードで、アクティビティを再開します。

- Backup.ShardedCluster.JournalCheckpoint()

調整されたジャーナル・チェックポイントを作成し、各データ・ノードを新しいジャーナル・ファイルに切り替えた後、チェックポイント番号とプリチェックポイント・ジャーナル・ファイルの名前を返します。プリチェックポイント・ファイルは、リストアに含める必要のある各データ・ノード上の最後のジャーナル・ファイルです。この後のジャーナル・ファイルには、チェックポイントで表される論理的な時点の後に生じたデータが含まれます。

- Backup.ShardedCluster.ExternalFreeze

  クラスタ全体の物理的なデータベース書き込みを凍結しますが、アプリケーションのアクティビティは凍結しません。その後、調整されたジャーナル・チェックポイントを作成し、各データ・ノードを新しいジャーナル・ファイルに切り替え、チェックポイント番号とプリチェックポイント・ジャーナル・ファイルの名前を返します。ExternalFreeze() の下に作成されたバックアップは、論理的な時点を表しませんが、プリチェックポイント・ジャーナル・ファイルが含まれているため、チェックポイントによって表される論理的な時点にリストアできます。

- Backup.ShardedCluster.ExternalThaw

  ExternalFreeze() によって一時停止されているディスク書き込みを再開します。

これらの呼び出しの技術情報に関するドキュメントについては、"インターシステムズ・クラス・リファレンス" を参照してください。

3.6.2.3 調整されたバックアップとリストアの手順

調整されたバックアップとリストアの 3 種類のアプローチに含まれる、シャーディング API を使用した手順については、以下のセクションで説明しています。

- 調整されたバックアップの作成

  一定期間、データベースのすべてのアクティビティを停止します。

- 調整されていないバックアップの作成後の、調整されたジャーナル・チェックポイント

  ダウンタイムが不要です。

- 調整されていないバックアップへの調整されたジャーナル・チェックポイントの追加

  ダウンタイムが不要です。

データ・ノードのバックアップには通常、データベース・ファイルだけでなく、InterSystems IRIS で使用されるすべてのファイル (ジャーナル・ディレクトリ、ライト・イメージ・ジャーナル、インストール・データ・ディレクトリなど) および必要な外部ファイルを含める必要があります。これらのファイルの場所は、一部にはクラスタの導入状況によって異なります ("シャード・クラスタの導入" を参照)。バックアップにこれらのファイルを含めるために必要な手法が、アプローチの選択に関わってくる可能性があります。

重要　ここで説明しているリストア手順では、リストア対象のデータ・ノードに使用可能なミラー・フェイルオーバー・パートナーがなく、災害復旧の状況でのみ、ミラーリングされたデータ・ノードで使用することが前提となっています。"ミラーリングされたシャード・クラスタの災害復旧"、および "高可用性ガイド" の "ミラー停止の手順" の章にある "災害復旧の手順" を参照してください。データ・ノードがミラーリングされている場合は、ミラーからプライマリを削除し、説明しているリストア手順を完了してから、"高可用性ガイド" の "ミラー・メンバの再構築" で説明されているようにミラー・メンバとして再構築します。

調整されたバックアップの作成

1. Backup.ShardedCluster.Quiesce を呼び出して、クラスタ内のすべてのデータ・ノード上のアクティビティ (およびこれによってすべてのアプリケーション・アクティビティ) を一時停止して、保留中のすべての書き込みがディスクにフラッシュされるまで待機します。このプロセスが完了して呼び出しが値を返すと、クラスタ内のすべてのデータベースおよびジャーナル・ファイルが論理的な同じ時点になります。
2. クラスタ内のすべてのデータ・ノードのバックアップを作成します。データベースのバックアップは調整されますが、開いているトランザクションが含まれる可能性があります。データ・ノードがバックアップからリストアされた後で再起動

されると、InterSystems IRIS のリカバリではジャーナル・ファイルを使用して、これらのトランザクションをロールバックすることで、トランザクションの整合性をリストアします。

3. バックアップが完了したら、Backup.ShardedCluster.Resume を呼び出して、データ・ノードの通常の動作をリストアします。

   重要　Resume() は、Quiesce() を呼び出したものと同じジョブ内で呼び出す必要があります。失敗の返り値は、Quiesce() の下に作成したバックアップ・イメージの信頼性が低く、このイメージの破棄が必要な可能性があることを示します。

4. 失敗の後に、各データ・ノードで以下の操作を行います。

   a. バックアップ・イメージをリストアします。

   b. 存在するジャーナル・ファイルは、バックアップの作成時点からリストアしたイメージに含まれるものだけであることを確認します。

      注意　これはきわめて重要です。起動時に、リカバリによってジャーナル・ファイルがリストアされ、バックアップの作成時点で開いていたトランザクションがすべてロールバックされるからです。バックアップの作成時点より後のジャーナル・データが起動時に存在する場合、そのジャーナル・データがリストアされ、対象のデータ・ノードが他のデータ・サーバと不整合になる可能性があります。

   c. 対象のデータ・ノードを再起動します。

   このデータ・ノードは、データベースのアクティビティが停止された論理的な時点にリストアされます。

注釈　この最初の 3 つの手順の代替方法として、クラスタ内のすべてのデータ・ノードを適切にシャットダウンして、コールド・バックアップを作成し、すべてのデータ・ノードを再起動することができます。

調整されていないバックアップの作成後の、調整されたジャーナル・チェックポイント

1. クラスタ内のすべてのデータ・ノードが動作中で、アプリケーション・アクティビティが続行している間に、これらのデータ・ノード上のデータベースのバックアップを作成します。これらのバックアップは、選択した任意の方法を使用して、選択した任意の間隔で、まったく異なる時点で作成できます。

2. Backup.ShardedCluster.JournalCheckpoint() を定期的に (スケジュールされたタスクとすることを推奨) 呼び出します。このメソッドは、調整されたジャーナル・チェックポイントを作成し、そのチェックポイントに達するために各データ・ノード上のリストアに含める最新のジャーナル・ファイルの名前を返します。データ・ノードを回復できる論理的な時点を指示するのは、バックアップのタイミングではなく、最新のチェックポイントの時点とプリチェックポイント・ジャーナル・ファイルの可用性であることに留意してください。

   注釈　ジャーナル・ファイルを切り替える前に、JournalCheckpoint() は、シャード・クラスタ内のすべてのデータ・ノードを短時間停止して、プリチェックポイント・ファイルすべてが論理的な同じ時点で終了するようにします。この結果、アプリケーション・アクティビティがこのメソッドの実行中にきわめて短い時間一時停止する可能性があります。

3. データ・ノードごとにジャーナル・ファイルの完全セットを必ず格納します。このセットには、最後のバックアップ時点から最新の調整されたジャーナル・チェックポイントの作成時点までのジャーナル・ファイルを格納し、最後がプリチェックポイント・ジャーナル・ファイルにします。また、これらのファイルすべてがサーバ障害発生後も必ず使用可能な状態にします (ジャーナル・ファイルを定期的にバックアップすることで実現する可能性が高い)。データベースのバックアップが調整されることはなく、部分トランザクションが存在する可能性もありますが、データ・ノードはバックアップからのリストア後に再起動され、調整されたジャーナル・ファイルをリカバリで使用して、すべてのデータベースを論理的な同じ時点にして、トランザクションの整合性をリストアします。

4. 障害発生後に、すべてのデータ・ノードに共通のリストア・ポイントとして利用可能な最新のチェックポイントを識別します。このためには、各データ・ノードに、最新のチェックポイントより前に作成されたデータベース・バックアップと、

プリチェックポイント・ジャーナル・ファイルまでの途中のジャーナル・ファイルすべてが格納されている必要があります。

注意　これはきわめて重要です。起動時に、リカバリによってジャーナル・ファイルがリストアされ、バックアップの作成時点で開いていたトランザクションがすべてロールバックされるからです。プリチェックポイント・ジャーナル・ファイルより後のジャーナル・ファイルが起動時に存在する場合、それらのジャーナル・ファイルがリストアされ、対象のデータ・ノードが他のデータ・サーバと不整合になる可能性があります。

5. 各データ・ノードで、チェックポイントより前のバックアップからデータベースをリストアし、ジャーナル・ファイルをチェックポイントまでリストアします。このチェックポイントの後のジャーナル・データが適用されることのないようにします。これを確実にする簡単な方法は、チェックポイントより後のジャーナル・ファイルがサーバに存在するかどうかを確認し、存在する場合はそれを移動または削除してから、ジャーナル・ログを削除することです。

   これで、調整されたジャーナル・チェックポイントが作成された論理的な時点にデータ・ノードがリストアされます。

調整されていないバックアップへの調整されたジャーナル・チェックポイントの追加

1. Backup.ShardedCluster.ExternalFreeze() を呼び出します。このメソッドは、シャード・クラスタ内のすべてのデータ・ノードのライト・デーモンを一時停止することで、それらのデータ・サーバ上のすべてのアクティビティを凍結します。アプリケーション・アクティビティは続行しますが、更新はジャーナル・ファイルだけに書き込まれ、ディスクにはコミットされません。このメソッドは、値を返す前に、調整されたジャーナル・チェックポイントを作成し、各データ・ノードを新しいジャーナル・ファイルに切り替えた後、チェックポイント番号とプリチェックポイント・ジャーナル・ファイルの名前を返します。この時点で、プリチェックポイント・ジャーナル・ファイルは、単一の論理的な時点を表します。

   注釈　Backup.ShardedCluster.ExternalFreeze() は内部的に Backup.ShardedCluster.JournalCheckpoint() を呼び出します。その結果、Backup.ShardedCluster.Quiesce() と Backup.ShardedCluster.Resume() が呼び出されてジャーナル・ファイルが切り替わる間、システムが短時間停止します。Resume() の実行が完了すると、`Backup.ShardedCluster.Resume: System resumed` メッセージがログに記録されます。これによってシステムが凍結しなくなるわけではなく、停止しなくなるにすぎません。ExternalFreeze() を呼び出すと、Backup.ShardedCluster.ExternalThaw() を呼び出すまでシステムは凍結した状態になります。

2. クラスタ内のすべてのデータ・ノードのバックアップを作成します。データベースのバックアップが調整されることはなく、部分トランザクションが存在する可能性もありますが、データ・ノードをリストアする際に必ず実行することは、データ・サーバをジャーナル・チェックポイントに回復し、すべてのデータベースを論理的な同じ時点にして、トランザクションの整合性をリストアすることです。

   注釈　Backup.ShardedCluster.ExternalFreeze() によってライト・デーモンが既定で 10 分間一時停止すると、アプリケーションのプロセスが以降の更新が行われないようブロックされます (ジャーナル・バッファがいっぱいになるリスクがあるため)。ただし、バックアップ・プロセスにもっと時間が必要な場合は、ExternalFreeze() のオプションの引数を使用してこの時間を延長できます。

3. すべてのバックアップが完了したら、Backup.ShardedCluster.ExternalThaw() を呼び出して、ライト・デーモンを再開し、データ・ノードの通常の動作をリストアします。

   重要　失敗の返り値は、ExternalFreeze() の下に作成したバックアップ・イメージの信頼性が低く、このイメージの破棄が必要な可能性があることを示します。

4. 失敗の後に、各データ・ノードで以下の操作を行います。

   a. バックアップ・イメージをリストアします。

   b. リストアされたイメージに、ExternalFreeze() から返されたプリチェックポイント・ジャーナル・ファイルより後のジャーナル・ファイルが存在する場合は、それらをすべて削除します。

c. InterSystems IRIS インスタンスを手動で回復するには、"データ整合性ガイド" の "バックアップとリストア" の章にある "自動 WIJ およびジャーナル・リカバリを使用しない InterSystems IRIS の起動" の手順に従います。ジャーナル・ファイルをリストアする場合、ExternalFreeze() によって切り替えられたジャーナル・ファイルから開始し、ExternalFreeze() によって返されたプリチェックポイント・ジャーナル・ファイルを最後にします(これらが同一のファイルである可能性があります。この場合、これがリストア対象の唯一のジャーナル・ファイルです)。

   注釈　コンテナ化された InterSystems IRIS インスタンスを使用して作業している場合に、コンテナ内で手動リカバリを実行する手順については、"コンテナ内でのインターシステムズ製品の実行" の "手動での起動が必要な場合のアップグレード" を参照してください。

調整されたジャーナル・チェックポイントが ExternalFreeze() メソッドによって作成された論理的な時点に、データ・ノードがリストアされます。

注釈　このアプローチでは、各データ・ノード上のデータベースとジャーナル・ファイルを、単一のバックアップにこれら両方を含めることができるように配置する必要があります。

3.6.3 ミラーリングされたシャード・クラスタの災害復旧

災害復旧 (DR) 非同期は、ミラーリングされたデータベースの同期された同じコピーをバックアップ・フェイルオーバー・メンバとして維持します。違いは、非同期とそのプライマリ間の通信が非同期であること、および非同期は自動フェイルオーバーに参加しないことです。ただし、フェイルオーバー・メンバの 1 つが利用できなくなった場合、DR 非同期をフェイルオーバー・メンバに昇格させてバックアップにすることができます。例えば、バックアップでメンテナンスを実行する場合や、プライマリの停止によってミラーがバックアップにフェイルオーバーしたときに、以前のプライマリの問題を調査・修正する一方で自動フェイルオーバー機能を維持しなければならない場合などです。重大な障害によって両方のフェイルオーバー・メンバの停止が発生した場合、昇格させた DR 非同期に手動でフェイルオーバーして災害復旧を行うことができます。

DR 非同期ミラー・メンバを使用すると、ミラーリングされたシャード・クラスタに災害復旧オプションを提供することが可能になり、ミラー・フェイルオーバー・ペアの停止後に比較的短時間でクラスタの動作を復旧できます。ミラーリングされたクラスタの災害復旧を可能にする場合、具体的には以下を行います。

- ミラーリングされたシャード・クラスタ内のすべてのデータ・ノード・ミラーで DR 非同期を少なくとも 1 つ構成します。

  重大な障害によってフェイルオーバー・ペアの停止が発生した場合に DR 非同期を確実に利用可能に維持するため、DR 非同期は通常、フェイルオーバー・ペアとは別のデータ・センターまたはアベイラビリティ・ゾーンに配置します。災害復旧手順で手動によってフェイルオーバーする先の DR 非同期が複数の場所に分散されている場合、DR 非同期間のネットワーク遅延がクラスタのパフォーマンスに大きな影響を及ぼす可能性があります。このため、ベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード・ミラーに、共通の場所にある DR 非同期を少なくとも 1 つ含めます。

  重要　災害復旧実現の一環として (または他の理由で) 既存のミラーリングされたクラスタ内のデータ・ノードに DR 非同期を追加する場合や、バックアップ・メンバを DR 非同期に降格させる場合は、いずれかのミラー・プライマリのクラスタ・ネームスペースで $SYSTEM.Sharding.VerifyShards() を呼び出してクラスタのメタデータを更新し、追加を反映させる必要があります。

- 前のセクションの説明に従って、調整されたバックアップを定期的に作成します。

  バックアップによってクラスタの動作がどの程度中断するか、バックアップの実行頻度、およびリストア手順の複雑さはすべて、選択した、調整されたバックアップとリストアのアプローチによって変わるため、アプローチを確認して、自身の状況と災害復旧の目標 (許容できるデータ損失の量、クラスタの動作をリストアする必要がある速度など) に最も適したアプローチを決定する必要があります。

- 必要な災害復旧手順 (選択した調整されたバックアップ手順について説明されているリストア手順を含む) を計画、準備、およびテストし、"高可用性ガイド" の "災害復旧の手順" で説明されている手順を十分に理解します。

必要な DR 非同期を構成済みで、調整されたバックアップを定期的に作成していることを前提とした場合、ミラーリングされたシャード・クラスタの一般的な災害復旧手順は以下のようになります。

1. 各データ・ノード・ミラーで以下を実行します。
   - "高可用性ガイド" の "DR 非同期メンバのフェイルオーバー・メンバへの昇格" で説明されている手順を使用して、DR 非同期 (他のすべてのデータ・ノード・ミラーの DR 非同期と共通の場所を共有しているものが理想的) をフェイルオーバー・メンバに昇格させます。
   - "災害復旧時の昇格した DR 非同期への手動フェイルオーバー" で説明されているように、昇格させた DR 非同期に手動でフェイルオーバーし、その DR 非同期をプライマリにします。
2. "調整されたバックアップとリストアの手順" の該当セクションで説明されているように、選択した調整されたバックアップとリストアのアプローチについて説明されている手順を使用して、最新の調整されたバックアップをリストアします。
3. フェイルオーバー機能をクラスタにリストアするために、各ミラーでフェイルオーバー・ペアを完成させます。データ・ノード・ミラーすべてに複数の DR 非同期が含まれていた場合、それぞれで別の DR 非同期をフェイルオーバー・メンバに昇格させます。ミラーに追加の DR 非同期がない場合は、"ミラー・データ・ノードによる高可用性" の説明に従って、各ミラーに 2 つ目のフェイルオーバー・メンバを構成します。
4. シャード・クラスタへのアプリケーション・アクセスをリストアします。

注釈　回復したミラーリングされたシャード・クラスタに計算ノードが含まれていた場合、これらの計算ノードはほぼ確実にデータ・ノードのフェイルオーバー・ペアと同じ場所に配置されており、障害によって利用できなくなっています。この場合、クラスタの完全なリカバリを行うには、"作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入" の説明に従って、回復したクラスタと同じ場所に配置された新しい計算ノードを導入して、ネットワーク遅延を最小化することが含まれます。クラスタの既存の計算ノードがまだ元の場所で動作している場合は、できる限り速やかにそれらの計算ノードをクラスタの新しい場所に再配置する必要があります。回復したクラスタは計算ノードなしでも動作可能ですが、計算ノードが提供する利点を利用することはできません。

3.6.4 シャード・クラスタの複製

現在のホストとは別の複数のホストに既存のクラスタを複製することが必要になる場合があります。その状況の例を以下に挙げます。

- プロダクション・クラスタのスナップショットに基づくテスト・システムを準備する。
- 既存ホストの一部またはすべてが使用できなくなる障害が発生した後、新しいホストにバックアップからクラスタを復元する。
- 現在とは別のデータ・センターなどの新しい場所にクラスタを移動する。

ここでは、新しいホストに既存のクラスタを複製して、上記のタスクまたは類似のタスクを達成できるようにする手順を説明します。

シャード・クラスタは、複数のマッピングと通信チャンネルを通じて連携するように構成したノードとネームスペースから成ります。この構成に関する情報は、データ・ノード 1 のマスタ・ネームスペース (または、ネームスペースレベル・クラスタのシャード・マスタ) に保存されて維持されます。この情報として、具体的なホスト名、ポート番号、ネームスペースなどがあります。クラスタを複製するには、この構成を新しいクラスタ上に複製する必要があります。

注釈　複製元の既存クラスタと複製先の新規クラスタでデータ・ノードの数が同じ場合にのみ、この手順を使用できます。

この複製プロセスには SYSTEM.Sharding API 呼び出しが関わっているので、複製されたクラスタは必ずネームスペースレベル・クラスタになります。このクラスタの管理と変更は、%SYSTEM.Sharding API および管理ポータルのネームスペース・レベルのページでのみ可能です。

この手順は、ホストあたりのノードが 1 つであることを想定していますが、ホストあたり複数のノードがあるネームスペースレベル・クラスタで使用できるように変更することもできます。

シャード・クラスタを複製するには以下の手順に従います。

1. 複製先クラスタのデータ・ノード・ホストをプロビジョニングまたは特定し、そのホストに InterSystems IRIS を導入します。管理ポータルまたは $SYSTEM.Sharding.EnableSharding API 呼び出しを使用してシャードを有効にします。
2. 複製元クラスタのデータ・ノード 1 にあるマスタ・ネームスペースで $SYSTEM.Sharding.GetConfig() を呼び出して、ノード 1 にあるマスタ・ネームスペースの既定のグローバル・データベースの場所と、各データ・ノードにあるシャード・ネームスペースの既定のグローバル・データベースの場所を表示します。新しいクラスタでは、元のクラスタのノードに対応するノードにこれらのデータベースを複製する必要があるので、必ず情報を正確に記録します。マスタ・ネームスペースで既定のルーチン・データベースとグローバル・データベースが異なる場合は、その情報も記録します。

   GetConfig() の出力には、各データ・ノードのシャード番号が記述されています。複製先のデータ・ノードを構成するには、複製元でそれに対応するデータ・ノードのシャード番号が必要です。複製元クラスタのシャード番号を複製先クラスタの各ホストに割り当てることによって、複製先ノードと複製元ノードとの対応関係を指定します。
3. 複製元クラスタへのアプリケーションのアクセスをブロックします。すべての書き込みをディスクにフラッシュした後、複製元クラスタにある InterSystems IRIS のすべてのインスタンスを 1 つずつシャット・ダウンします。
4. 複製先クラスタ上に複製元クラスタのグローバル・データベースを複製します (必要に応じて、マスタのルーチン・データベースも複製します)。この作業には以下の 2 つの方法のいずれかを使用します。
   - 複製元クラスタでデータベースをバックアップし、それを複製先クラスタに復元する。
   - 複製元クラスタから複製先クラスタに IRIS.DAT ファイルをコピーする。

   この 2 つのクラスタでディレクトリ・パスを同じにする必要はありません。どちらの方法を使用する場合でも、必ず複製元のシャード番号に対応する複製先クラスタ・ノード上に各データベースを複製します。

   注釈　複製元クラスタをミラーリングしている場合は、各ノードのいずれか 1 つのフェイルオーバー・メンバのみからデータベースをバックアップまたはコピーするだけですみます。
5. 複製先クラスタの各ノードでシャード・ネームスペースを作成し、複製したシャード・データベースをそのネームスペースの既定のグローバル・データベースとします。ノード 1 で、マスタ・ネームスペースに対して上記と同じ手順を実行します。ルーチン・データベースとグローバル・データベースが異なる場合はルーチン・データベースを追加します。
6. 複製先クラスタのノード 1 にあるマスタ・ネームスペースで以下の手順に従います。
   a. 複製先データ・ノートごとに $SYSTEM.Sharding.ReassignShard() を 1 回ずつ呼び出し、それぞれのホスト名または IP アドレス、スーパーサーバ・ポート、シャード・ネームスペース、シャード番号を指定します。複製元クラスタに対応する正しいシャード番号を指定します。以下に例を挙げます。

      ```
      set status =
      $SYSTEM.Sharding.ReassignShard(,"shard-host",shard-port,"shard-namespace","shard-number")
      ```

   b. $SYSTEM.Sharding.Reinitialize() を呼び出します。これによって、複製先クラスタのバージョンに互換性があることが確認され、複製先クラスタの有効化に必要なマッピング、ECP 接続、メタデータがすべて設定されます。$SYSTEM.Sharding.VerifyShards() が自動的に呼び出されて、クラスタが有効になり、構成が正しいことが確認されます。

複製先クラスタに複製したグローバル・データベースとルーチン・データベース以外のデータベースに対して、ユーザ定義のグローバル、ルーチン、またはパッケージのマッピングがマスタ・ネームスペースにあると、Reinitialize() からエラーが返されます。新しいクラスタにはこのような追加のデータベースが存在しないからです。このエラーを回避するには、Reinitialize() を呼び出すときの 2 番目の引数 `IgnoreMappings` に、以下のように `1` を指定します。

```
set status = $SYSTEM.Sharding.Reinitialize(,1)
```

Reinitialize() 呼び出しの後、グローバル・データベースとルーチン・データベースを複製したときと同様に、関連する追加のデータベースを複製先ノード 1 に複製します。これらのデータベースをすべて用意した段階で $SYSTEM.Sharding.VerifyShards() を呼び出し、関連するマッピングをシャード・ネームスペースに適用します。

7. 必要に応じて、複製したクラスタをミラーリングに変換し、計算ノードを追加します。

3.6.5 シャーディング API

このリリースで、InterSystems IRIS は、シャード・クラスタの構成および管理に使用できる 2 つの API を用意しています。

- %SYSTEM.Cluster API は、現在のアーキテクチャの導入および管理に使用します (″シャーディングの要素″ を参照)。
- %SYSTEM.Sharding API は、以前のバージョンのネームスペース・レベルのアーキテクチャの導入および管理に使用します (″ ネームスペース・レベルのシャーディング・アーキテクチャ″ を参照)。

3.6.5.1 %SYSTEM.Cluster API

%SYSTEM.Cluster API メソッドおよびそれらを呼び出す手順の詳細は、″インターシステムズ・クラス・リファレンス″ の ″%SYSTEM.Cluster″ クラス・ドキュメントを参照してください。

%SYSTEM.Cluster API メソッドを以下の要領で使用します。

- InterSystems IRIS インスタンスを新しいシャード・クラスタの最初のノードとして設定するには、Initialize を呼び出します。
- インスタンスをデータ・ノードとしてクラスタに追加するには、追加されるインスタンスで AttachAsDataNode を呼び出します。
- インスタンスを計算ノードとしてクラスタに追加するには、追加されるインスタンスで AttachAsComputeNode を呼び出します。
- クラスタのノードのリストを表示するには、ListNodes を呼び出します。
- クラスタのメタデータの概要を取得するには、GetMetaData を呼び出します。
- 現在のインスタンスのクラスタ・ネームスペースの名前を取得するには、ClusterNamespace を呼び出します。

%SYSTEM.Cluster メソッドには以下のようなものがあります。メソッドの名前をクリックすると、そのクラス・リファレンス・エントリが開きます。

- $SYSTEM.Cluster.Initialize()

  現在の InterSystems IRIS インスタンスをクラスタの最初のノードとして有効にするのに必要なすべての手順を、自動的かつ透過的に実行します。

- $SYSTEM.Cluster.AttachAsDataNode()

  現在の InterSystems IRIS インスタンスを指定したクラスタにデータ・ノードとしてアタッチします。

- $SYSTEM.Cluster.AttachAsComputeNode()

現在の InterSystems IRIS インスタンスを指定したクラスタに計算ノードとしてアタッチします。

- $SYSTEM.Cluster.ListNodes()

  現在の InterSystems IRIS インスタンスが属するクラスタのノードを、コンソールまたは指定した出力ファイルにリストします。

- $SYSTEM.Cluster.GetMetaData()

  現在の InterSystems IRIS インスタンスが属するクラスタのメタデータの概要を取得します。

- $SYSTEM.Cluster.ClusterNamespace()

  現在の InterSystems IRIS インスタンス上のクラスタ・ネームスペースの名前を取得します。

3.6.5.2 %SYSTEM.Sharding API

%SYSTEM.Sharding API メソッドおよびそれらを呼び出す手順の詳細は、"インターシステムズ・クラス・リファレンス" の "%SYSTEM.Sharding" クラス・ドキュメントを参照してください。

%SYSTEM.Sharding API メソッドを以下の要領で使用します。

- InterSystems IRIS インスタンスがシャード・マスタまたはシャード・サーバとして機能できるようにするには、Enable-Sharding メソッドを呼び出します。
- マスタ・ネームスペースに属するシャードを定義します。そのためには、マスタ・ネームスペースで AssignShard を呼び出す操作をシャードごとに 1 回ずつ実行します。
- シャードが割り当てられたら、VerifyShards を呼び出して、これらに到達可能であること、およびこれらが正しく構成されていることを検証します。
- シャード・テーブルが既に格納されているネームスペースに追加のシャードが割り当てられ、AssignShard の呼び出し時に自動検証のために新しいシャードに到達できなかった場合、到達可能になった後、ActivateNewShards を呼び出して、それらを有効にすることができます。
- マスタ・ネームスペースに割り当てられたすべてのシャードをリストするには、ListShards を呼び出します。
- 既存の各データ・ノード上にミラーを作成した後に、ミラーリングされていないクラスタをミラーリングされたクラスタに変換する場合、AddDatabasesToMirrors を呼び出して、それぞれのミラーにマスタ・データベースとシャード・データベースを追加します。
- データ・ノード/シャード・データ・サーバの追加後、既存のシャード・データをクラスタ全体で再分散するには、$SYSTEM.Sharding.Rebalance() を使用します ("データ・ノードの追加とデータの再分散" を参照)。
- シャード・データ・サーバを別のアドレスにある別のシャード・ネームスペースに割り当てるには、ReassignShard を呼び出します。
- マスタ・ネームスペースに属するセットからシャードを削除するには、DeassignShard を呼び出します。
- シャーディング構成オプションを設定するには、SetOption を呼び出し、それらの現在の値を取得するには、GetOption を呼び出します。

%SYSTEM.Sharding メソッドには以下のようなものがあります。メソッドの名前をクリックすると、そのクラス・リファレンス・エントリが開きます。

- $SYSTEM.Sharding.EnableSharding()

  現在の InterSystems IRIS インスタンスがシャード・マスタまたはシャード・サーバとして機能できるようにします。

- $SYSTEM.Sharding.AssignShard()

  シャードをマスタ・ネームスペースに割り当てます。

- $SYSTEM.Sharding.VerifyShards()

割り当てられたシャードが到達可能であり、正しく構成されているかどうかを検証します。

- $SYSTEM.Sharding.ListShards()

  指定されたマスタ・ネームスペースに割り当てられているシャードをコンソールまたは現在のデバイスにリストします。

- $SYSTEM.Sharding.ActivateNewShards()

  前に AssignShard を呼び出したときに有効にすることができなかったシャードを有効にします。

- $SYSTEM.Sharding.AddDatabasesToMirrors()

  ミラー・フェイルオーバー・メンバとして追加されたデータ・ノードのマスタ・データベースとシャード・データベースをそのそれぞれのミラーに追加します。

- $SYSTEM.Sharding.Rebalance()

  データ・ノード/シャード・データ・サーバの追加後、クラスタ全体で既存のシャード・データを再分散します。

- $SYSTEM.Sharding.ReassignShard()

  異なるアドレスの異なるシャード・ネームスペースにシャード番号を割り当てて、シャードを割り当て直します。

- $SYSTEM.Sharding.DeassignShard()

  前に割り当てられていたマスタ・ネームスペースからシャードを割り当て解除します。これにより、マスタ・ネームスペースに属するシャードのセットからそのシャードが削除されます。

- $SYSTEM.Sharding.SetOption()

  指定されたマスタ・ネームスペースの範囲内で、指定されたシャーディング構成オプションを指定の値に設定します。

- $SYSTEM.Sharding.GetOption()

  指定されたマスタ・ネームスペースの範囲内で、指定されたシャーディング構成オプションの値を取得します。

- $SYSTEM.Sharding.SetNodeIPAddress()

  クラスタ通信のためのアドレスとして、ノードのホスト名ではなく、指定された IP アドレスを構成します (すべてのノードで使用する必要があります)。

- $SYSTEM.Sharding.GetConfig()

  指定されたマスタ・ネームスペースまたはクラスタ・ネームスペースが属するシャード・クラスタに関する構成情報を取得します。

- $SYSTEM.Sharding.Reinitialize()

  複製されたシャード・クラスタ (″シャード・クラスタの複製″ を参照) の内部的なマッピング、接続、クラスタ・メタデータを再初期化します。

- $SYSTEM.Sharding.GetFederatedTableInfo()

  指定したフェデレーション・テーブルに関する情報を返します。

- $SYSTEM.Sharding.Help()

  %SYSTEM.Sharding のメソッドの概要を表示します。

3.6.6 ネームスペース・レベル・アーキテクチャの導入

管理ポータルまたは %SYSTEM.Sharding を使用して、シャード・マスタ、シャード・データ・サーバ、およびオプションのクエリ・サーバで構成される、古いネームスペース・レベル・アーキテクチャで InterSystems IRIS シャード・クラスタを導入

するには、このセクションの手順を使用します。これらの手順は、各 InterSystems IRIS インスタンスを独自のシステムにインストールすることを前提としています。

アプリケーションは任意のシャード・データ・サーバのクラスタ・ネームスペースに接続して、全データセットをローカルのものであるかのように利用できるので、シャード・クエリ・サーバを含まないネームスペース・レベルのクラスタの一般的な推奨ベスト・プラクティスは、すべてのシャード・データ・サーバ間でアプリケーション接続を負荷分散することです。一般的に、これはシャード・クエリ・サーバを含むクラスタにも最適なアプローチですが、追加の考慮事項があります。詳細は、"計算ノードの計画" のセクションの末尾を参照してください。

ノード・レベルのクラスタのノード・レベルの導入と異なり、ネームスペース・レベルの導入では、クラスタ構成の一部としてミラーを作成することはできません。ただし、ミラーリングされたシャード・クラスタ (つまり、ミラーリングされたシャード・マスタ・データ・サーバとミラーリングされたシャード・データ・サーバで構成) を導入する場合は、以下のどれでも実行できます。

- 既存のミラーのプライマリをシャード・マスタ・データ・サーバとして構成し、最初に対象のマスタ・ネームスペースのグローバル・データベースをミラーに追加する。
- 既存のシャード・マスタ・データ・サーバをプライマリとして含むミラーを作成し、バックアップを追加する前に、マスタ・ネームスペースのグローバル・データベースをミラーに追加する。
- 既存のミラーのフェイルオーバー・メンバをシャード・マスタ・データ・サーバとして割り当て、最初に対象のクラスタ・ネームスペースのグローバル・データベースをミラーに追加する。
- 割り当てたシャード・データ・サーバをプライマリとして含むミラーを作成し、シャード・ネームスペースのグローバル・データベースをミラーに追加してから、バックアップに関する情報を指定して、ミラーリングされたシャード・データ・サーバとして再割り当てする。

ミラーリングされたシャード・クラスタに DR 非同期を含めることはできますが、レポート非同期を含めることはできません。推奨されるベスト・プラクティスは、ミラーリングされたノードとミラーリングされていないノードを混在させないようにすることです。つまり、シャード・マスタとすべてのシャード・データ・サーバをミラーリングするか、いずれもミラーリングしないようにしてください。これらの手順のステップを以下に示します。このステップは、ミラーリングされたクラスタを導入する場合にも、ミラーリングされていないクラスタを導入する場合にも使用できます。

- インフラストラクチャのプロビジョニングまたは特定
- クラスタ・ノードへの InterSystems IRIS の導入
- シャード・データ・サーバの準備
- 以下のいずれかを使用したクラスタの構成
  - 管理ポータル
  - %SYSTEM.Sharding API

3.6.6.1 インフラストラクチャのプロビジョニングまたは特定

必要なネットワーク・ホスト・システム (物理、仮想、またはクラウド) の数を特定します (シャード・マスタおよびシャード・データ・サーバについてそれぞれ 1 つのホスト)。

ミラー・クラスタを導入する場合は、シャード・マスタ・データ・サーバ用に 2 つのホストと、各シャード・データ・サーバ用に 2 つ以上のホスト (DR 非同期メンバが必要かどうかによる) をプロビジョニングします。

重要　必ず、"%SYSTEM.Cluster API を使用したクラスタの導入" の "インフラストラクチャのプロビジョニングまたは特定" で、シャード・クラスタのインフラストラクチャの要件とベスト・プラクティスを確認してください。

3.6.6.2 クラスタ・ノードへの InterSystems IRIS の導入

この手順では、各システムで単一の InterSystems IRIS インスタンスをホストするか、またはホストする予定であることが前提となっています。

1. インターシステムズが提供するイメージからコンテナを作成する (″コンテナ内でのインターシステムズ製品の実行″ を参照) か、キットから InterSystems IRIS をインストールする (″インストール・ガイド″ を参照) ことにより、InterSystems IRIS のインスタンスを導入します。

   重要　必ず、″%SYSTEM.Cluster API を使用したクラスタの導入″ の ″データ・ノードへの InterSystems IRIS の導入″ で、シャード・クラスタの InterSystems IRIS インスタンスの要件とベスト・プラクティスを確認してください。

2. ″データベース・キャッシュとデータベースのサイズの見積もり″ の説明に従って、各インスタンスのデータベースをホストするストレージ・デバイスがグローバル・データベースのターゲット・サイズに十分対応できる大きさであることを確認します。

   可能であれば、すべてのインスタンスについて、データベース・ディレクトリとジャーナル・ディレクトリを別個のストレージ・デバイスに配置してください。これは、大量のデータ取り込みがクエリの実行と同時に行われる場合に特に重要です。ファイル・システムの構成およびジャーナル・ストレージなどのストレージの構成のガイドラインについては、″システム・リソースの計画と管理″ の ″ストレージの計画″ と ″ファイル・システムの分離″、および ″データ整合性ガイド″ の ″ジャーナリングの最善の使用方法″ を参照してください。

3. ″データベース・キャッシュとデータベースのサイズの見積もり″ で決定したサイズに従って、クラスタにおける最終的なロールに応じて、各インスタンスのデータベース・キャッシュ (グローバル・バッファ・プール) を割り当てます。データベース・キャッシュを割り当てる手順については、″システム管理ガイド″ の “InterSystems IRIS の構成” の章にある ″メモリと開始設定″ を参照してください。

   注釈　クラスタ・メンバの共有メモリ・ヒープのサイズを大きくすることがお勧めの手段となることがあります。共有メモリ・ヒープへのメモリの割り当て方法の詳細は、″構成パラメータ・ファイル・リファレンス″ の ″gmheap″ を参照してください。

   InterSystems IRIS インスタンスのルーチン・キャッシュとデータベース・キャッシュおよび共有メモリ・ヒープにメモリを割り当てる際のガイドラインは、″システム・リソースの計画と管理″ を参照してください。

3.6.6.3 管理ポータルを使用したネームスペース・レベルのクラスタの構成

インフラストラクチャをプロビジョニングまたは特定して、クラスタ・ノードに InterSystems IRIS をインストールした後、このセクションのステップに従って、管理ポータルを使用してネームスペース・レベルのクラスタを構成します。

ブラウザで管理ポータルを開く方法の詳細は、コンテナに導入されたインスタンスの手順、または ″InterSystems IRIS の基礎 : IDE の接続″ の ″InterSystems IRIS 接続情報″ のセクションで、キットからインストールしたインスタンスの手順を参照してください。

シャード・データ・サーバの準備

目的の各インスタンスで以下を実行して、クラスタにシャード・データ・サーバとして追加するための準備を行います。

注釈　状況によっては、API (管理ポータルの基盤) で 1 つ以上のノードのホスト名をクラスタのノードの相互接続に使用できる IP アドレスに解決できない場合があります。このような場合、$SYSTEM.Sharding.SetNodeIPAddress() ("%SYSTEM.Sharding API" を参照) を呼び出して、各ノードで使用する IP アドレスを指定できます。$SYSTEM.Sharding.SetNodeIPAddress() を使用するには、対象の各クラスタ・ノードでこれを呼び出してから、これらのノードで他の %SYSTEM.Sharding API 呼び出しを行う必要があります。以下に例を示します。下線

```
set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
```

この呼び出しを使用する場合、シャードをマスタ・ネームスペースに割り当てる際に、ホスト名ではなく、各シャード・データ・サーバ・ノードに対して指定した IP アドレスを使用する必要があります。"シャード・マスタ・データ・サーバの構成とシャード・データ・サーバの割り当て" を参照してください。

1. インスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→** [Sharding]→**[シャード有効]** の順に選択し、表示されるダイアログで **[OK]** をクリックします(既定値はほぼすべてのクラスタに適しているため、**[ECP 接続の最大数]** 設定の値を変更する必要はありません)。
2. インスタンスを再起動します(管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動した後に再読み込みするだけでかまいません)。
3. シャード・ネームスペースを作成するには、[Configure Namespace-Level] ページ (**[システム管理]→[構成]→[システム構成]→**[Sharding]**→**[Configure Namespace-Level]) に移動して、**[ネームスペースを作成]** ボタンをクリックします。続いて、"システム管理ガイド" の "InterSystems IRIS の構成" の章にある "ネームスペースの作成/変更" の手順に従います (ネームスペースは相互運用対応である必要はありません)。別のシャード・データ・サーバのシャード・ネームスペースには異なる名前を付けることができますが、同じ名前を付けた方が便利です。

   このプロセスでは、"データベース・キャッシュとデータベースのサイズの見積もり" で説明しているように、既定のグローバル・データベースで新しいデータベースを作成し、必ず、シャード・ネームスペースのターゲット・サイズに対応できる十分な空き容量があるデバイスにそのデータベースを配置してください。データ取り込みのパフォーマンスが重要な考慮事項である場合は、データベースの初期サイズをそのターゲット・サイズに設定します。また、作成したグローバル・データベースを、ネームスペースの既定のルーチン・データベースとして選択します。

   重要　シャード・データ・サーバにするために既存のミラーを準備する場合は、各メンバのシャード・ネームスペースの既定のグローバル・データベースを作成する際に (最初にプライマリ、次にバックアップ、その次に DR 非同期)、**[ミラー・データベース?]** プロンプトで **[はい]** を選択して、シャード・ネームスペースのグローバル・データベースをミラーに含めます。

   注釈　"データベース・キャッシュとデータベースのサイズの見積もり" で説明したように、シャード・マスタ・データ・サーバとシャード・データ・サーバはすべて、単一の既定のグローバル・データベースを共有し、このデータベースは、シャード・マスタに物理的に配置され、マスタ・グローバル・データベースと呼ばれます。シャード・ネームスペースが作成されたときに作成される既定のグローバル・データベースはシャードに残りますが、シャードに保存されているデータを含む local globals database になります。シャード・データ・サーバは、クラスタに割り当てられるまでマスタ・グローバル・データベースを使用しません。このため、わかりやすくするために、このドキュメントの計画ガイドラインと手順では、最終的なローカル・グローバル・データベースを、シャード・ネームスペースの既定のグローバル・データベースと呼んでいます。

   新しいネームスペースは、**IRISTEMP** が一時ストレージ・データベースとして構成された状態で自動的に作成されます。シャード・ネームスペースのこの設定を変更しないでください。
4. 後の手順のために、ホスト・システムの DNS 名または IP アドレス、インスタンスのスーパーサーバ (TCP) ポート、および作成したシャード・ネームスペースの名前を記録します。

注釈　もう 1 つのノード (この手順で必要なノード) から見ると、コンテナ化された InterSystems IRIS インスタンスのスーパーサーバ・ポートは、そのコンテナが作成されたときにスーパーサーバ・ポートが公開されたホスト・ポートによって異なります。この詳細および例は、"コンテナ内でのインターシステムズ製品の実行" の "永続的な %SYS を使用した InterSystems IRIS コンテナの実行" と "InterSystems IRIS コンテナの実行 : Docker Compose の例"、および Docker ドキュメントの "Container networking" を参照してください。

そのホスト上にしかないコンテナ化されていない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。インスタンスのスーパーサーバ・ポート番号を表示または設定するには、そのインスタンスの管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します (インスタンスの管理ポータルを開いてスーパーサーバ・ポートを確認する方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎 : IDE の接続" の "InterSystems IRIS 接続情報" で、キットからインストールしたインスタンスの手順を参照してください)。

シャード・マスタ・データ・サーバの構成とシャード・データ・サーバの割り当て

目的のインスタンスで以下を実行してシャード・マスタ・データ・サーバとして構成し、準備しておいたシャード・データ・サーバをクラスタに割り当てます。この手順を使用してクラスタにクエリ・シャードを割り当てることもできます。

1. インスタンスの管理ポータルを開き、**[システム管理]→[構成]→[システム構成]→** [Sharding]**→[シャード有効]** の順に選択し、表示されるダイアログで **[OK]** をクリックします(既定値はほぼすべてのクラスタに適しているため、**[ECP 接続の最大数]** 設定の値を変更する必要はありません)。
2. インスタンスを再起動します(管理ポータルが表示されているブラウザのウィンドウやタブを閉じる必要はありません。インスタンスが完全に再起動した後に再読み込みするだけでかまいません)。
3. マスタ・ネームスペースを作成するには、[Configure Namespace-Level] ページ (**[システム管理]→[構成]→[システム構成]→[Sharding]→[Configure Namespace-Level]**) に移動して、**[ネームスペースを作成]** ボタンをクリックします。続いて、"システム管理ガイド" の "InterSystems IRIS の構成" の章にある "ネームスペースの作成/変更" の手順に従います (ネームスペースは相互運用対応である必要はありません)。

   このプロセスでは、"データベース・キャッシュとデータベースのサイズの見積もり" で説明しているように、既定のグローバル・データベースで新しいデータベースを作成し、必ず、マスタ・ネームスペースのターゲット・サイズに対応できる十分な空き容量があるデバイスにそのデータベースを配置してください。また、作成したグローバル・データベースを、ネームスペースの既定のルーチン・データベースとして選択します。

   重要　既存のミラーをシャード・マスタ・データ・サーバとして構成する場合は、各メンバのマスタ・ネームスペースの既定のグローバル・データベースを作成する際に (最初にプライマリ、次にバックアップ、その次に DR 非同期)、**[ミラー・データベース?]** プロンプトで **[はい]** を選択して、マスタ・ネームスペースのグローバル・データベースをミラーに含めます。

4. シャード・データ・サーバを割り当てるには、**[ネームスペース]** ドロップダウンが先ほど作成したマスタ・ネームスペースに設定されていることを確認します。各シャードに対して、[Configure Namespace-Level] ページの **[シャード割り当て]** ボタンをクリックして [シャード割り当て] ダイアログを表示し、シャード・データ・サーバ・インスタンスの以下の情報を入力します。
   - ホストの名前 (または、このセクションの冒頭で説明したように、この手順を開始する前に各ノードで $SYSTEM.Sharding.SetNodeIPAddress() 呼び出しを使用した場合は IP アドレス)。
   - スーパーサーバ・ポート。
   - 準備の際に作成したシャード・ネームスペースの名前。
   - シャードのロールとして **[データ]** を選択。

     重要　シャード・クエリ・サーバを割り当てるには、シャードのロールとして、**[データ]** ではなく **[クエリ]** を選択します。

既存のミラーをシャード・データ・サーバとして割り当てる場合は、プライマリから作業を始め、上記の情報を入力した後に、**[ミラー]** を選択して以下の情報を指定します。これにより、バックアップが自動的にプライマリに割り当てられます(シャード・クエリ・サーバはミラーリングされません)。

- ミラーの名前。
- バックアップのホストの名前 (または IP アドレス)。
- バックアップのスーパーサーバ・ポート。
- ミラーの仮想 IP アドレス (VIP) (設定されている場合)。

最後に **[完了]** を選択します。

5. シャードを割り当てると、そのシャードは [Configure Namespace-Level] ページのリストに表示されます。シャードがミラーである場合は、両方のフェイルオーバー・メンバが含まれます。
6. すべてのシャード・データ・サーバ (およびオプションでシャード・クエリ・サーバ) を割り当てたら、**[シャードを検証]** をクリックして、ポートが正しく、ノードの必要な情報がすべて設定されていて、シャード・マスタがシャード・データ・サーバと通信できることを確認します。

シャード・サーバの割り当て解除

一覧表示されているシャード・データ・サーバまたはシャード・クエリ・サーバの右側にある **[割り当て解除]** リンクを使用して、シャード・サーバをクラスタから削除できます。ミラーリングされたシャード・データ・サーバを割り当て解除する場合は、プライマリの管理ポータルで割り当て解除を行う必要があります。

重要　クラスタにシャード・テーブルまたはクラスがある場合は (テーブルにデータが含まれるかどうかは関係ありません)、シャード・データ・サーバを割り当て解除することはできません。

3.6.6.4 API を使用したクラスタ・ノードの構成

インフラストラクチャをプロビジョニングまたは特定して、クラスタ・ノードに InterSystems IRIS をインストールした後、このセクションのステップに従って、API を使用してネームスペース・レベルのクラスタを構成します。ここで説明されている呼び出し、および API の他の呼び出しの詳細は、"InterSystems クラス・リファレンス" の %SYSTEM.Sharding のクラス・ドキュメントを参照してください。

注釈　状況によっては、API で 1 つ以上のノードのホスト名をクラスタのノードの相互接続に使用できる IP アドレスに解決できない場合があります。このような場合、$SYSTEM.Sharding.SetNodeIPAddress() ("%SYSTEM.Sharding API" を参照) を呼び出して、各ノードで使用する IP アドレスを指定できます。$SYSTEM.Sharding.SetNodeIPAddress() を使用するには、対象の各クラスタ・ノードでこれを呼び出してから、これらのノードで他の %SYSTEM.Sharding API 呼び出しを行う必要があります。以下に例を示します。下線

```
set status = $SYSTEM.Sharding.SetNodeIPAddress("00.53.183.209")
```

この呼び出しを使用する場合、シャード・マスタで $SYSTEM.Sharding.AssignShard() を呼び出してクラスタにノードを割り当てる際、以下の手順に示すように、shard-host 引数として、ホスト名ではなく、各ノードに対して指定した IP アドレスを使用する必要があります。

シャード・データ・サーバの準備

目的の各インスタンスで以下を実行して、クラスタにシャード・データ・サーバとして追加するための準備を行います。

注釈　ネームスペース・レベルの導入でミラーを作成することはできません。ただし、既存のミラーのフェイルオーバー・メンバをシャード・データ・サーバとして追加できます。このためには、ミラーを作成し、この手順を使用してメンバを準備してから、次の手順の説明に従ってメンバをシャード・データ・サーバとして割り当てます。

1. "システム管理ガイド" の "InterSystems IRIS の構成" の章にある "ネームスペースの作成/変更" の説明に従って、管理ポータルを使用してシャード・ネームスペースを作成します。(ネームスペースは相互運用対応である必要はありません。)別のシャード・データ・サーバのシャード・ネームスペースには異なる名前を付けることができますが、同じ名前を付けた方が便利です。

   "データベース・キャッシュとデータベースのサイズの見積もり" で説明しているように、新しいデータベースを既定のグローバル・データベースとして作成し、必ず、そのターゲット・サイズに対応できる十分な空き容量があるデバイスにそのデータベースを配置してください。データ取り込みのパフォーマンスが重要な考慮事項である場合は、データベースの初期サイズをそのターゲット・サイズに設定します。

   重要　シャード・データ・サーバにするために既存のミラーを準備する場合は、各メンバのシャード・ネームスペースの既定のグローバル・データベースを作成する際に(最初にプライマリ、次にバックアップ、その次に DR 非同期)、**[ミラー・データベース?]** プロンプトで **[はい]** を選択して、シャード・ネームスペースのグローバル・データベースをミラーに含めます。

   作成したグローバル・データベースを、ネームスペースの既定のルーチン・データベースとして選択します。

   注釈　"データベース・キャッシュとデータベースのサイズの見積もり" で説明したように、シャード・マスタ・データ・サーバとシャード・データ・サーバはすべて、単一の既定のグローバル・データベースを共有し、このデータベースは、シャード・マスタに物理的に配置され、マスタ・グローバル・データベースと呼ばれます。シャード・ネームスペースが作成されたときに作成される既定のグローバル・データベースはシャードに残りますが、シャードに保存されているデータを含む **local globals database** になります。シャード・データ・サーバは、クラスタに割り当てられるまでマスタ・グローバル・データベースを使用しません。このため、わかりやすくするために、このドキュメントの計画ガイドラインと手順では、最終的なローカル・グローバル・データベースを、シャード・ネームスペースの既定のグローバル・データベースと呼んでいます。

   新しいネームスペースは、**IRISTEMP** が一時ストレージ・データベースとして構成された状態で自動的に作成されます。シャード・ネームスペースのこの設定を変更しないでください。

2. 後の手順のために、ホスト・システムの DNS 名または IP アドレス、インスタンスのスーパーサーバ (TCP) ポート、および作成したシャード・ネームスペースの名前を記録します。

   注釈　もう 1 つのノード (この手順で必要なノード) から見ると、コンテナ化された InterSystems IRIS インスタンスのスーパーサーバ・ポートは、そのコンテナが作成されたときにスーパーサーバ・ポートが公開されたホスト・ポートによって異なります。この詳細および例は、"コンテナ内でのインターシステムズ製品の実行" の "永続的な %SYS を使用した InterSystems IRIS コンテナの実行" と "InterSystems IRIS コンテナの実行 : Docker Compose の例"、および Docker ドキュメントの "Container networking" を参照してください。

   そのホスト上にしかないコンテナ化されていない InterSystems IRIS インスタンスの既定のスーパーサーバ・ポート番号は 1972 です。インスタンスのスーパーサーバ・ポート番号を表示または設定するには、そのインスタンスの管理ポータルで **[システム管理]→[構成]→[システム構成]→[メモリと開始設定]** を選択します (インスタンスの管理ポータルを開いてスーパーサーバ・ポートを確認する方法の詳細は、コンテナに導入されたインスタンスの手順、または "InterSystems IRIS の基礎 : IDE の接続" の "InterSystems IRIS 接続情報" で、キットからインストールしたインスタンスの手順を参照してください)。

3. InterSystems ターミナル・ウィンドウで、任意のネームスペースで $SYSTEM.Sharding.EnableSharding を呼び出して ("%SYSTEM.Sharding API" を参照)、インスタンスがシャード・クラスタに参加できるようにします。次に例を示します。

   ```
   set status = $SYSTEM.Sharding.EnableSharding()
   ```

   引数は必要ありません。

注釈　これらの手順で詳述されている各 API 呼び出しの戻り値（成功の場合は 1 など）を確認するには、以下を入力します。

```
zw status
```

状況によっては通知なしで呼び出しが失敗することがあるため、各呼び出しの後に **[ステータス]** を確認することをお勧めします。呼び出しが成功しなかった（**[ステータス]** が [1] 以外）場合、以下を入力することにより、わかりやすいエラー・メッセージが表示されます。

```
do $SYSTEM.Status.DisplayError(status)
```

この呼び出しを実行した後で、インスタンスを再起動します。

シャード・マスタ・データ・サーバの構成とシャード・データ・サーバの割り当て

シャード・マスタ・データ・サーバのインスタンスで、以下を実行してシャード・マスタを構成し、シャード・データ・サーバを割り当てます。

1. ″システム管理ガイド″ の “InterSystems IRIS の構成” の章にある ″ネームスペースの作成/変更″ の説明に従って、管理ポータルを使用してマスタ・ネームスペースを作成します。（ネームスペースは相互運用対応である必要はありません。）

   ″データベース・キャッシュとデータベースのサイズの見積もり″ で説明しているように、必ず、作成する既定のグローバル・データベースをそのターゲット・サイズに対応できる十分な空き容量があるデバイスに配置してください。データ取り込みのパフォーマンスが重要な考慮事項である場合は、データベースの初期サイズをそのターゲット・サイズに設定します。

   重要　既存のミラーをシャード・マスタ・データ・サーバとして構成する場合は、各メンバのマスタ・ネームスペースの既定のグローバル・データベースを作成する際に（最初にプライマリ、次にバックアップ、その次に DR 非同期）、**[ミラー・データベース?]** プロンプトで **[はい]** を選択して、マスタ・ネームスペースのグローバル・データベースをミラーに含めます。

   作成したグローバル・データベースを、ネームスペースの既定のルーチン・データベースとして選択します。

   注釈　新しいネームスペースは、**IRISTEMP** が一時ストレージ・データベースとして構成された状態で自動的に作成されます。マスタ・ネームスペースのこの設定を変更しないでください。シャード・クエリの中間結果が IRISTEMP に格納されるため、このデータベースは、拡張に対応できる十分な空き容量がある、使用可能な最速のストレージに配置する必要があります。大規模な結果セットを返す多くの同時シャード・クエリが予想される場合は特に、その必要があります。

2. InterSystems ターミナル・ウィンドウで、任意のネームスペースで以下の手順を実行します。

   a. $SYSTEM.Sharding.EnableSharding() を呼び出して（″%SYSTEM.Sharding API″ を参照）、インスタンスがシャード・クラスタに参加できるようにします（引数は必要ありません）。次に例を示します。

   ```
   set status = $SYSTEM.Sharding.EnableSharding()
   ```

   この呼び出しを実行した後で、インスタンスを再起動します。

   b. シャード・データ・サーバごとに 1 回ずつ $SYSTEM.Sharding.AssignShard() を呼び出して（″%SYSTEM.Sharding API″ を参照）、作成したマスタ・ネームスペースにシャードを割り当てます。次に例を示します。

   ```
   set status = $SYSTEM.Sharding.AssignShard("master-namespace","shard-host",shard-superserver-port,
       "shard_namespace")
   ```

引数は、作成したマスタ・ネームスペースの名前と、前の手順でそのシャード・データ・サーバについて記録した情報を表します。次に例を示します。

```
set status = $SYSTEM.Sharding.AssignShard("master","shardserver3",1972,"shard3")
```

このセクションの冒頭で説明したように、この手順を始める前に各ノードで $SYSTEM.Sharding.SetNodeIPAddress() 呼び出しを使用した場合、shard-host には、ホスト名ではなくシャード・ホストの IP アドレスを使用してください。

c. シャードを正しく割り当てているかどうかを検証するには、以下のコマンドを発行し、ホスト、ポート、およびネームスペースの名前を確認します。

```
do $SYSTEM.Sharding.ListShards()
Shard   Host                        Port    Namespc  Mirror  Role    VIP
1       shard1.internal.acme.com    56775   SHARD1
2       shard2.internal.acme.com    56777   SHARD2
...
```

注釈　InterSystems IRIS インスタンスのスーパーサーバ・ポートの特定についての重要な情報は、"シャード・データ・サーバの構成" の手順 2 を参照してください。

d. シャード・マスタがシャード・データ・サーバと通信できるように、ポートが正しく、ノードの必要な構成がすべて適切であることを確認するには、次のように $SYSTEM.Sharding.VerifyShards() ("%SYSTEM.Sharding API" を参照) を呼び出します。

```
do $SYSTEM.Sharding.VerifyShards()
```

$SYSTEM.Sharding.VerifyShards() 呼び出しは、多数のエラーを識別します。例えば、$SYSTEM.Sharding.AssignShard() 呼び出しで指定されたポートが、シャード・データ・サーバ・ホスト上で開いているポートであり、InterSystems IRIS インスタンスのスーパーサーバ・ポートではない場合、シャードは正しく割り当てられません。$SYSTEM.Sharding.VerifyShards() 呼び出しはこれを示します。

e. ミラーのフェイルオーバー・ペアをシャード・データ・サーバとして準備および追加した場合は、以下に示すように、シャード・マスタ・データ・サーバで $SYSTEM.Sharding.AddDatabasesToMirrors() 呼び出し ("%SYSTEM.Sharding API" を参照) を使用して、各プライマリにマスタ・データベースとシャード・データベースをミラーリングされたデータベースとして追加し、各フェイルオーバー・ペアをミラーリングされたシャード・データ・サーバとして完全に構成できます。

```
set status = $SYSTEM.Sharding.AddDatabasesToMirrors("master-namespace")
```

推奨されるベスト・プラクティスは、ミラーリングされたノードとミラーリングされていないノードを混在させるのではなく、完全にミラーリングするか、まったくミラーリングしないでクラスタを導入することです。

構成したデータ・ノード・ミラー内のフェイルオーバー・ペアに DR 非同期を追加するには、最初のフェイルオーバー・メンバ上のミラーリングされたデータベースに対応する新しいノード上にデータベースを作成します。続いて、そのミラーに新しいノードを DR 非同期として追加し、最初のフェイルオーバー・メンバ上に作成したバックアップからデータベースをリストアして、そのデータベースを自動的にミラーに追加します。

既存のデータ・ノードそれぞれにミラーを作成してから、ノード 1 で $SYSTEM.Sharding.AddDatabasesToMirrors() を呼び出して、クラスタをミラーリングされた構成に自動的に変換します。

3.6.7 予約名

以下の名前は InterSystems IRIS で使用されているため、ユーザ定義要素の名前には使用しないでください。

- パッケージ名 **IRIS.Shard** は、システム生成シャード・ローカル・クラス名用に予約されているため、ユーザ定義クラスには使用しないでください。

- スキーマ名 IRIS_Shard は、システム生成シャード・ローカル・テーブル名用に予約されているため、ユーザ定義テーブルには使用しないでください。
- プレフィックス IRIS.Shard.、IS.、および BfVY. は、シャード・ローカル・テーブルのグローバル用に予約されており、シャード・ネームスペースでシャードのローカル・データベースにマッピングされます。ユーザ定義グローバル名およびシャード化されていないテーブルのグローバル名は、これらのプレフィックスで始めないでください。シャード・ローカル・テーブル以外のグローバルにこれらのプレフィックスを使用すると、予測できない動作が起こる可能性があります。