



- 1. 改訂情報
- 2. intra-mart Accel Platform でのバージョンアップ、アップデートの概要
 - 2.1. intra-mart Accel Platform へのバージョンアップ、アップデートの流れ
 - 2.1.1. intra-mart BaseModule Ver.5.0 から intra-mart WebPlatform/AppFramework Ver.7.1 ま
でを intra-mart Accel Platform へバージョンアップする。
 - 2.1.2. intra-mart WebPlatform/AppFramework Ver.7.2 を intra-mart Accel Platform へバージョ
ンアップする。
 - 2.1.3. intra-mart Accel Platform 2012 Winter(Bourbon) から 2013 Autumn(Eden) まです、
intra-mart Accel Platform の最新版 へアップデートする。
 - 2.1.4. intra-mart Accel Platform 2013 Winter(Felicia) を、intra-mart Accel Platform の 最新版
へアップデートする。
 - 2.1.5. intra-mart Accel Platform 2014 Spring(Granada) 以降を、intra-mart Accel Platform の最
新版へアップデートする。
- 3. Rhino のバージョンアップにおける非互換事項
 - 3.1. intra-mart Accel Platform 2026 Spring(Mimosa) 以前から 2026 Autumn(Nemesia) 以降へアッ
プデートする場合の Rhino の非互換事項
 - 3.1.1. 概要
 - 3.1.2. 影響箇所の確認方法
 - 3.1.3. アップデート時に必要な作業
 - 3.1.4. スクリプト開発モデル で互換性が担保できない変更点
 - 3.1.5. IM-LogicDesigner のユーザ定義(Javascript)タスクで互換性が担保できない変更点
 - 3.1.6. Java API (モジュール開発者向け)

変更年月日	変更内容
2014-08-01	初版
2026-10-01	第2版 下記を追加しました。 ▪ 「Rhino のバージョンアップにおける非互換事項」を追加。

intra-mart Accel Platform — バージョンアップ、アップデートの手引き 第2版 2026-10-01

intra-mart Accel Platform でのバージョンアップ、アップデートの概要

項目

- intra-mart Accel Platform へのバージョンアップ、アップデートの流れ
 - intra-mart BaseModule Ver.5.0 から intra-mart WebPlatform/AppFramework Ver.7.1 までを intra-mart Accel Platform へバージョンアップする。
 - 手順
 - intra-mart WebPlatform/AppFramework Ver.7.2 を intra-mart Accel Platform へバージョンアップする。
 - 手順
 - intra-mart Accel Platform 2012 Winter(Bourbon) から 2013 Autumn(Eden) までを、intra-mart Accel Platform の最新版 へアップデートする。
 - 手順
 - intra-mart Accel Platform 2013 Winter(Felicia) を、intra-mart Accel Platform の 最新版 へアップデートする。
 - 手順
 - intra-mart Accel Platform 2014 Spring(Granada) 以降を、intra-mart Accel Platform の最新版 へアップデートする。
 - 手順

intra-mart Accel Platform へのバージョンアップ、アップデートの流れ

旧バージョンの intra-mart WebPlatform/AppFramework から intra-mart Accel Platform へのバージョンアップ、や intra-mart Accel Platform の既存の環境から、最新版へアップデートの流れは、下図のような流れになります。各製品、バージョン、アップデート毎で、下記図中の 1 から 5 のパターンに分けられ、各々手順が異なりますので、以下を参考に、詳細は各ドキュメントを参照の上、実施してください。

../_images/verup-update.png

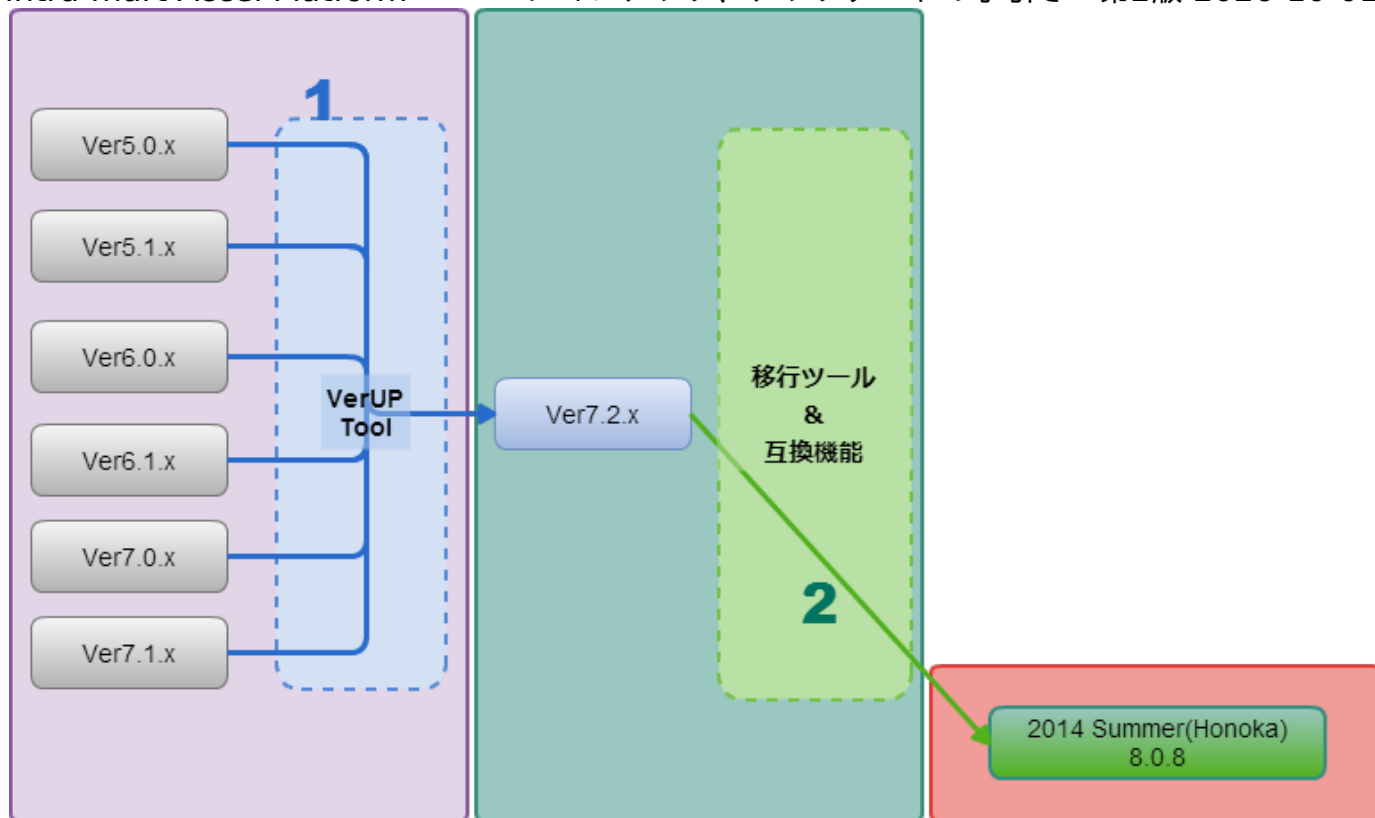


注意

- 「3」, 「4」, 「5」に該当する Accel製品のアップデートは、「**プロダクト毎に行う作業**」も確認してください。

intra-mart BaseModule Ver.5.0 から intra-mart WebPlatform/AppFramework Ver.7.1 までを intra-mart Accel Platform へバージョンアップする。

上記図上の1に相当する、intra-mart BaseModule Ver.5.0 から intra-mart WebPlatform/AppFramework Ver.7.1 までをご利用の環境を、intra-mart Accel Platform へバージョンアップする場合、一度、intra-mart WebPlatform/AppFramework Ver.7.2 にバージョンアップする必要があります。現在利用の環境を、intra-mart WebPlatform/AppFramework Ver.7.2 にバージョンアップするには、VerUPツールを利用することで、一気に Ver.7.2 にバージョンアップすることが可能です。



コラム

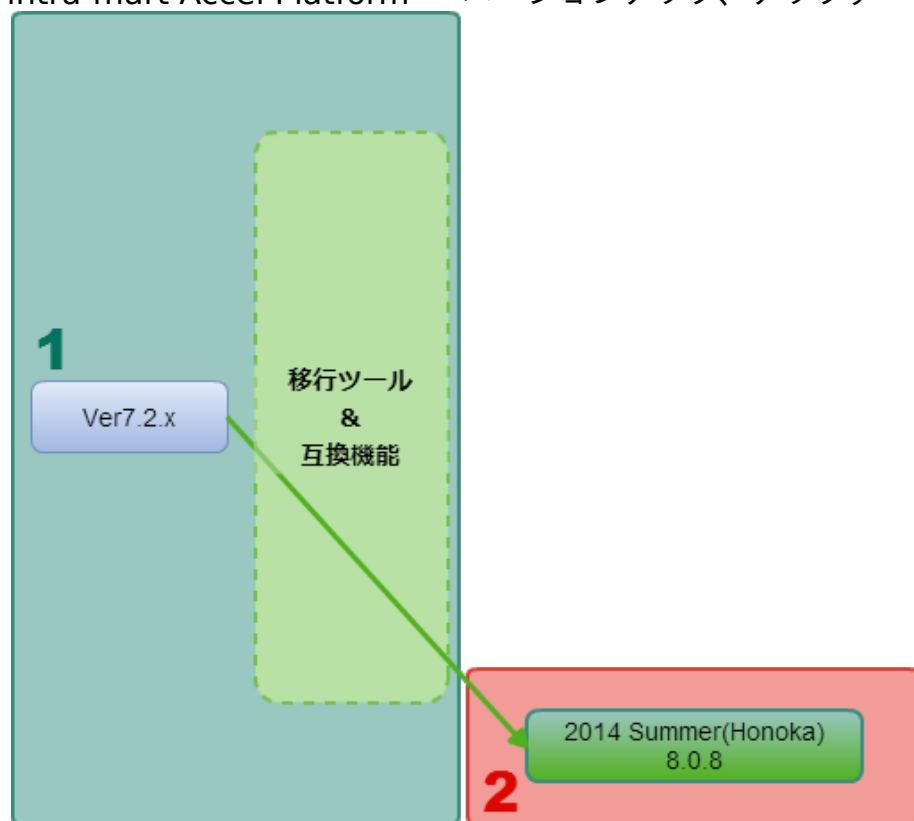
- VerUPツールについては、弊社営業または、intra-mart パートナーにお問い合わせください。

手順

- VerUPツールにて、intra-mart WebPlatform/AppFramework Ver.7.2 にバージョンアップする。
 - この際、構築する intra-mart WebPlatform/AppFramework Ver.7.2 には、必ず、IM-共通マスタのインストールとセットアップを行ってください。
- IM-Juggling にて、intra-mart Accel Platform の互換機能、移行ツールを含んだWARファイルを作成し、テナント環境セットアップを実行する。
 - 移行作業の詳細は、
 - 移行ガイド (Version7.2 から Accel Platform 2014 Springへ移行する場合)
 - 移行ガイド (Version7.2 から Accel Platform 2013 Winter以前へ移行する場合)
 を参照してください。

intra-mart WebPlatform/AppFramework Ver.7.2 を intra-mart Accel Platform へバージョンアップする。

上記図上の2に相当する、intra-mart WebPlatform/AppFramework Ver.7.2 にてご利用の環境を、intra-mart Accel Platform へバージョンアップする場合は、intra-mart Accel Platform の互換機能と移行ツールを含んだ環境を構築し、テナント環境セットアップを実行することで、intra-mart WebPlatform/AppFramework Ver.7.2 の環境をintra-mart Accel Platform にバージョンアップすることができます。

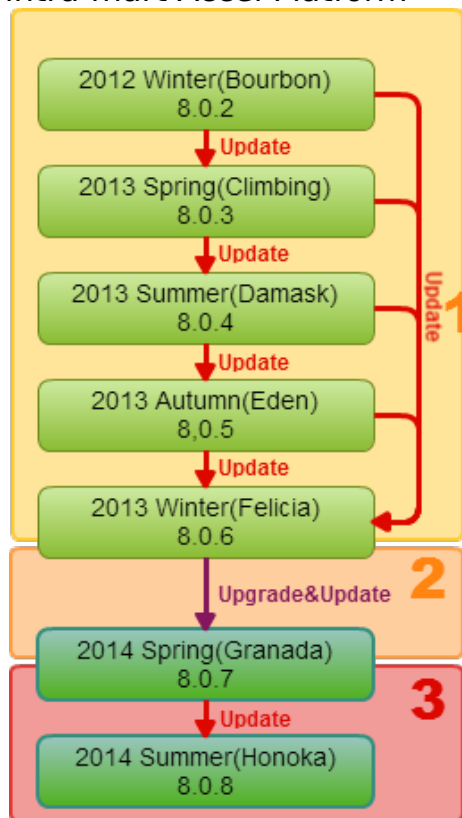


手順

1. VerUp対象の intra-mart WebPlatform/AppFramework Ver.7.2 に、IM-共通マスタがインストールされていない場合は、必ず、IM-共通マスタのインストールとセットアップを行ってください。
2. IM-Juggling にて、intra-mart Accel Platform の互換機能、移行ツールを含んだWARファイルを作成し、テナント環境セットアップを実行する。
 - 移行作業の詳細は、[移行ガイド](#) を参照してください。

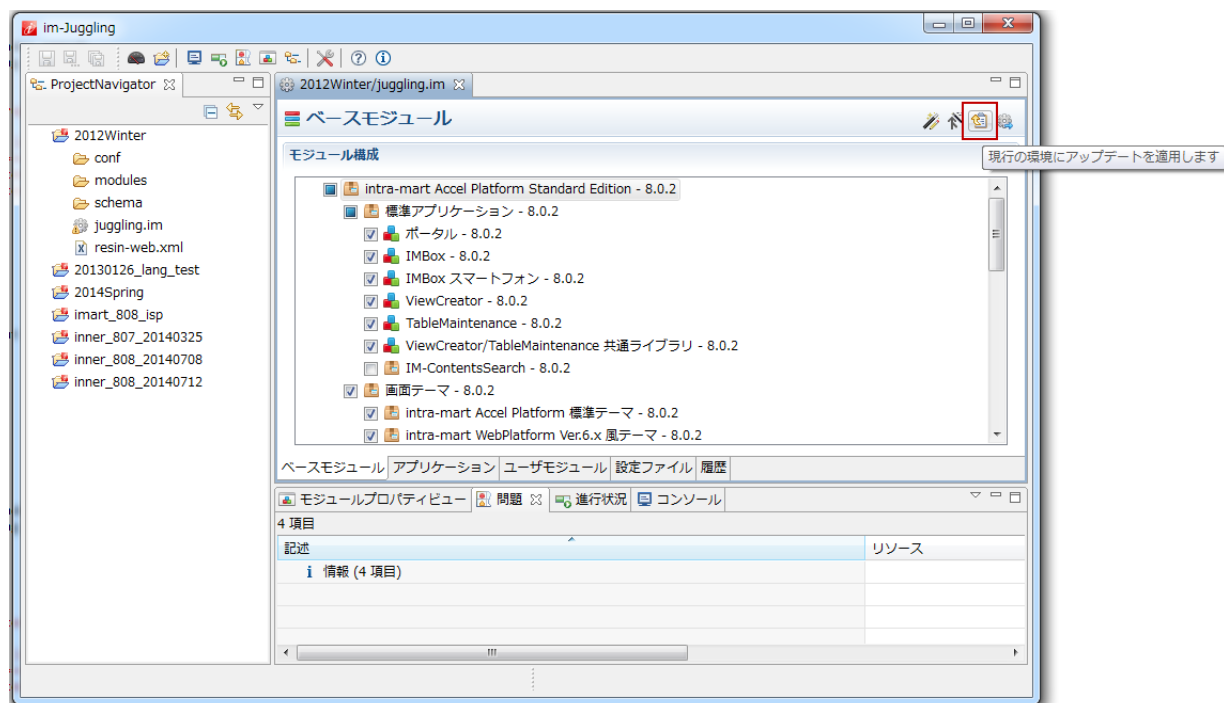
intra-mart Accel Platform 2012 Winter(Bourbon) から 2013 Autumn(Eden) までを、intra-mart Accel Platform の最新版 へアップデートする。

上記図上の3に相当する、intra-mart Accel Platform 2012 Winter(Bourbon) から 2013 Autumn(Eden) までを、intra-mart Accel Platform の 最新版 へアップデートする場合は、一度、2013 Winter(Felicia) にアップデートし、2014 Spring(Granada) にアップデート後にその後に最新版にアップデートすることになります。

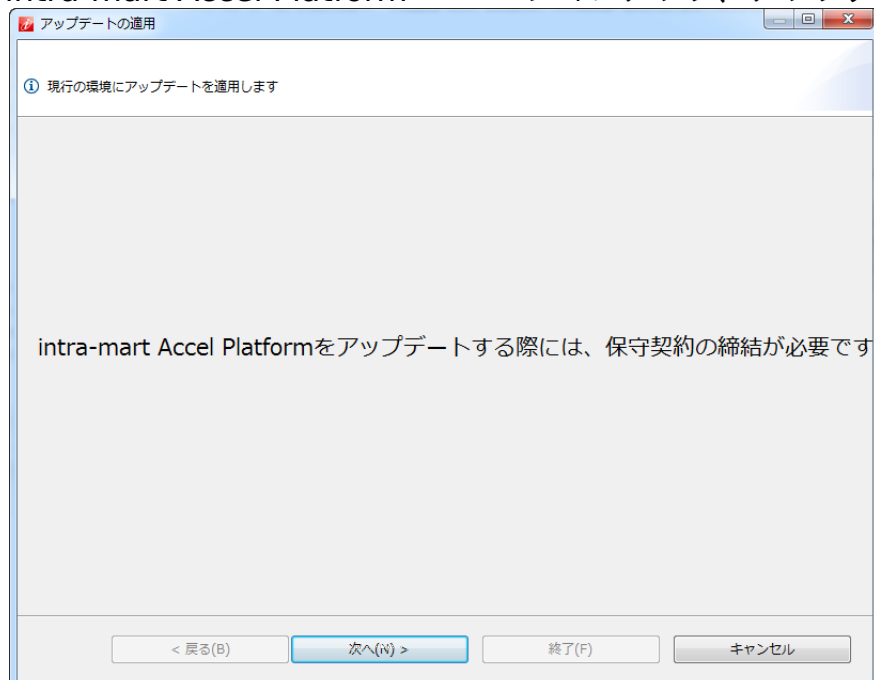


手順

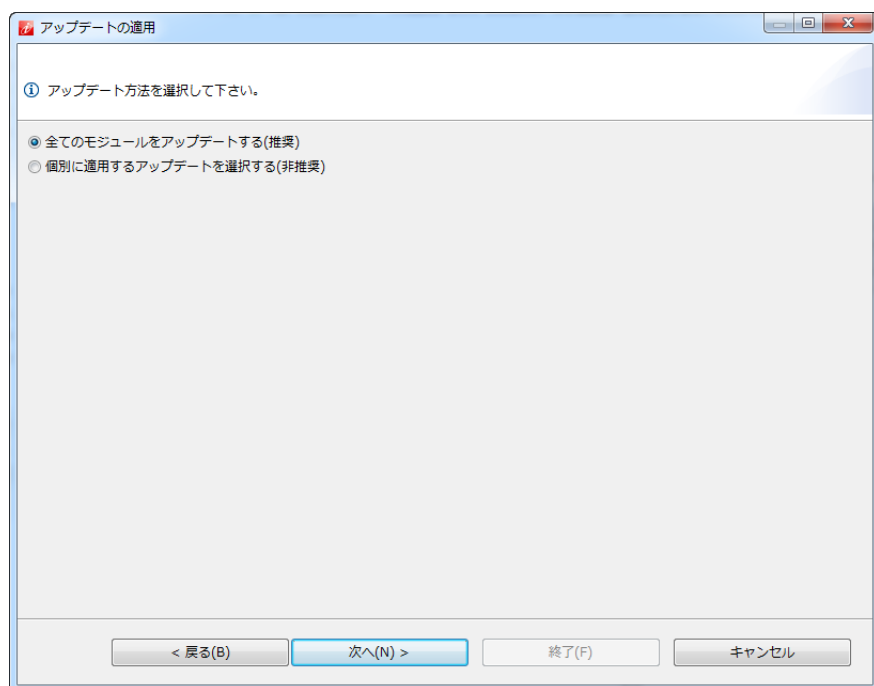
1. IM-Juggling にて、既存の環境に対して、2013 Winter(Felicia) (8.0.6) へのアップデートを行ってください。
 - 1.1. IM-Juggling 上で、既存のプロジェクトを開き、以下のように、右上の「アップデート」をクリックしてください。



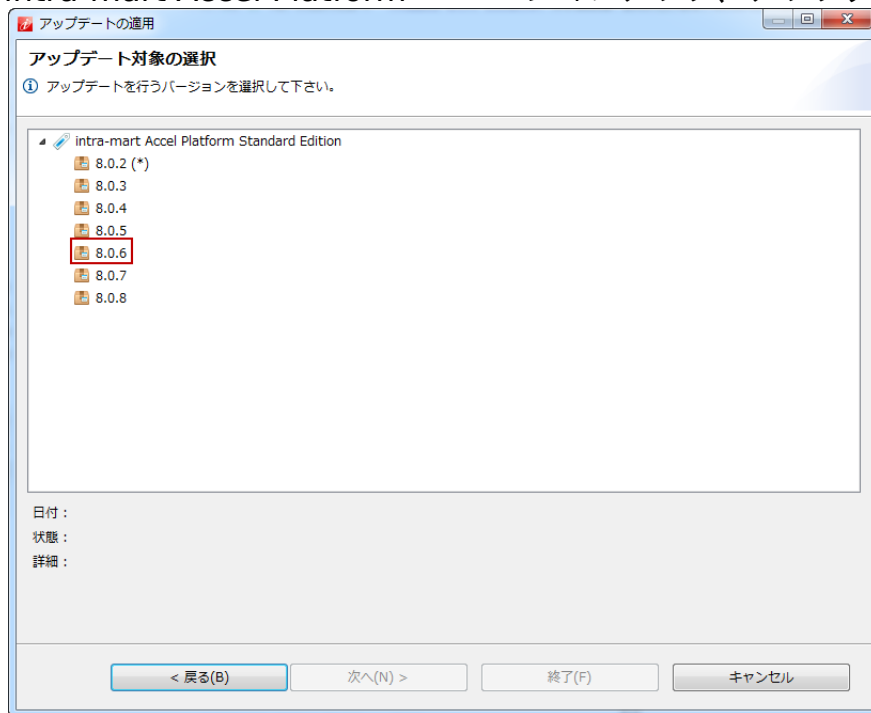
- 1.2. 「保守契約の締結が必要」という警告が表示されますので、保守契約が締結されている場合は「次へ」をクリックし、保守契約が締結されていない場合は、「キャンセル」で終了し、保守契約を締結してください。



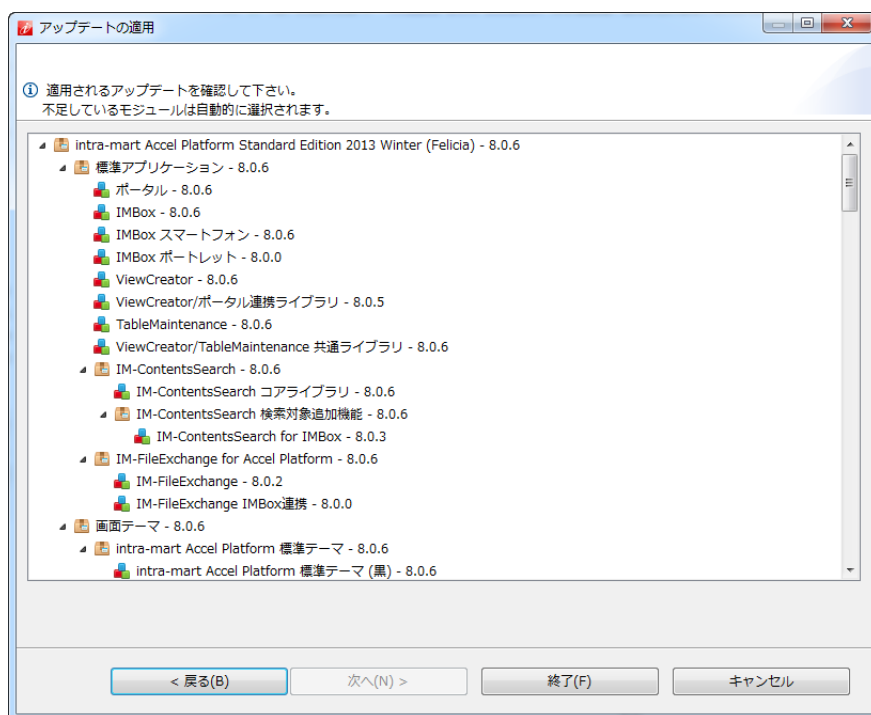
1.3. 「アップデート方法」の選択が表示されますので、選択して「次へ」をクリックしてください。



1.4. 「アップデートを行うバージョン」の選択が表示されますので、「8.0.6」を選択して「次へ」をクリックしてください。



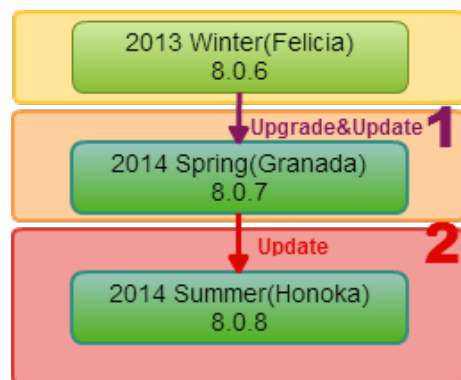
1.5. 「適用されるアップデート確認」が表示されますので、問題がなければ、「終了」をクリックすることで、既存のプロジェクトに対してアップデートが実行されます。



2. アップデート終了後、WARファイルを作成し、アプリケーションサーバにデプロイしてください。
3. デプロイ後、アプリケーションサーバを起動し、システム管理画面にて、テナント環境セットアップを実行してください。
4. iAP 806 -> iAP 807 アップデートツールの実行。
 - アップデートガイド(2013 Winter から 2014 Spring) を参照して、2013 Winter から 2014 Springにアップデートしてください。
5. 2014 Spring にアップデートしたプロジェクトに対して、再度、IM-Juggling にて、最新版へのアップデートを行ってください。

intra-mart Accel Platform 2013 Winter(Felicia) を、intra-mart Accel Platform の 最新版 へアップデートする。

上記図上の4に相当する、intra-mart Accel Platform 2012 Winter(Bourbon) を、intra-mart Accel Platform の最新版へアップデートする場合は、一度、2014 Spring(Granada) にアップデートし、その後に最新版にアップデートすることになります。



手順

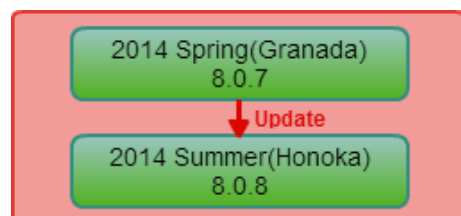
1. iAP 806 -> iAP 807 アップデートツールの実行。

- アップデートガイド(2013 Winter から 2014 Spring) を参照して、2013 Winter から 2014 Springにアップデートしてください。

2. 2014 Spring にアップデートしたプロジェクトに対して、再度、IM-Juggling にて、最新版へのアップデートを行ってください。

intra-mart Accel Platform 2014 Spring(Granada) 以降を、intra-mart Accel Platform の最新版へアップデートする。

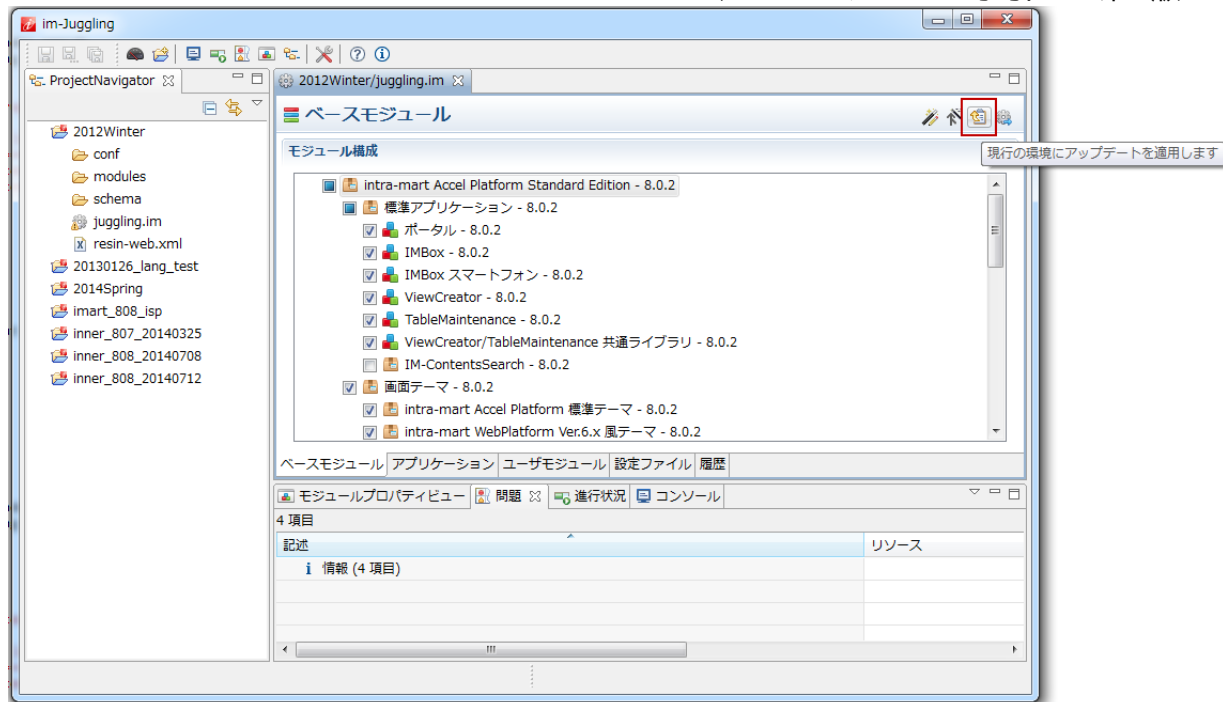
上記図上の5に相当する、intra-mart Accel Platform 2014 Spring(Granada) 以降を、intra-mart Accel Platform の最新版へアップデートする場合は、IM-Juggling にて最新版にアップデートすることになります。



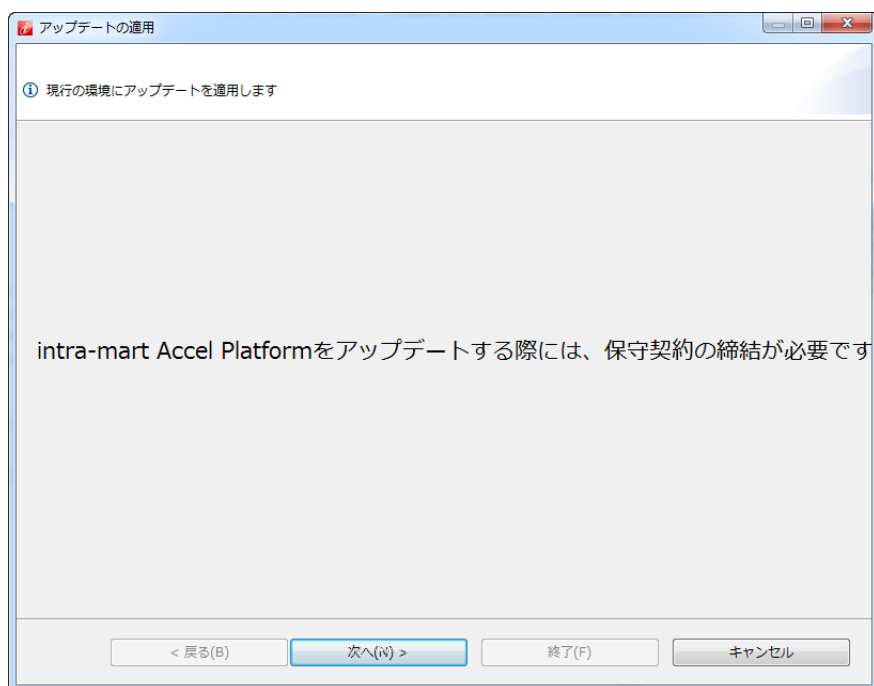
手順

1. IM-Juggling にて、既存の環境に対して、最新版 へのアップデートを行ってください。

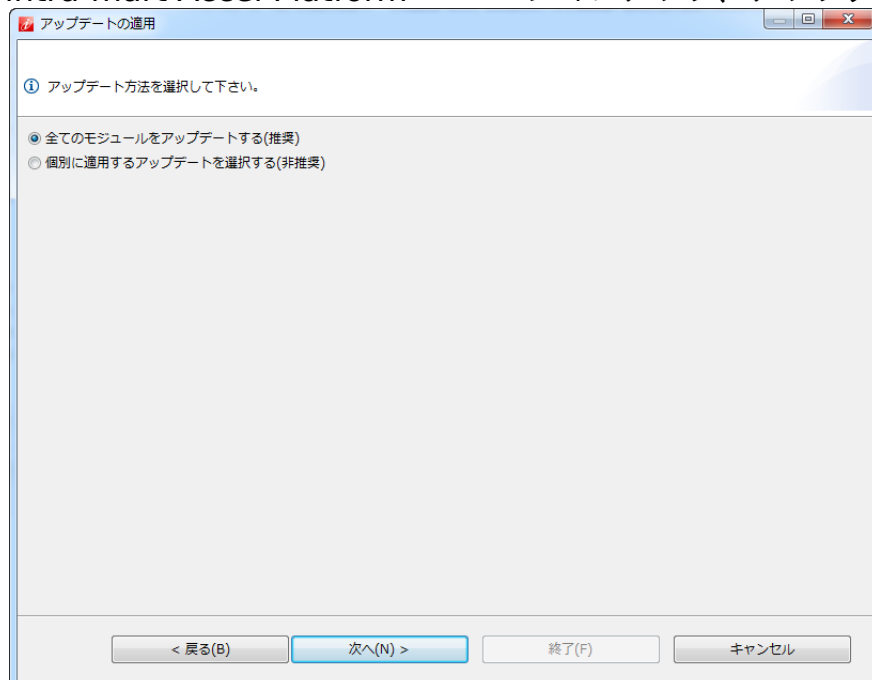
- 1.1 IM-Juggling 上で、既存のプロジェクトを開き、以下のように、右上の「アップデート」をクリックしてください。



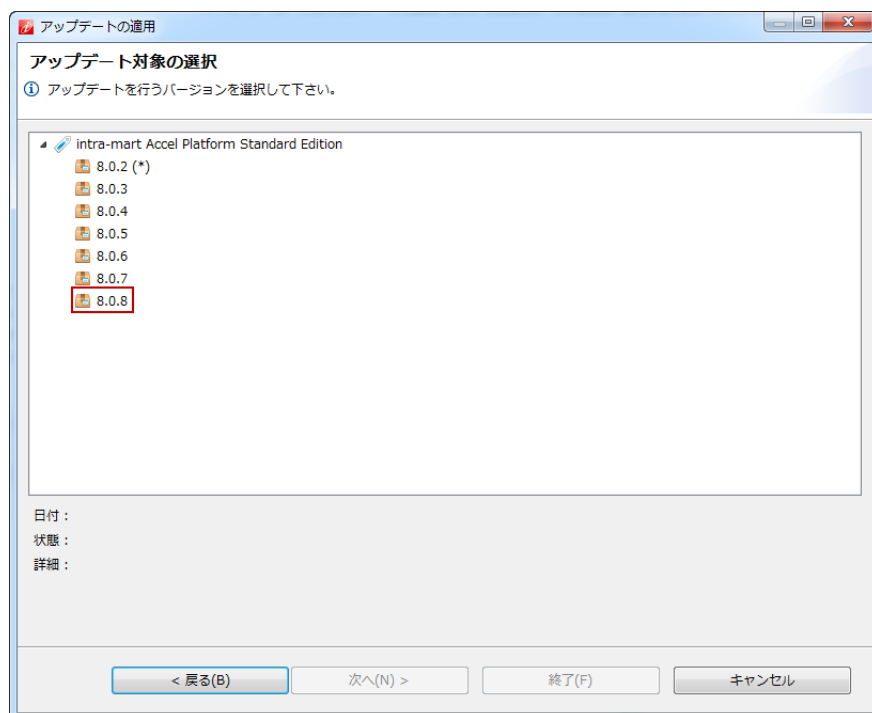
1.2. 「保守契約の締結が必要」という警告が表示されますので、保守契約が締結されている場合は「次へ」をクリックし、保守契約が締結されていない場合は、「キャンセル」で終了し、保守契約を締結してください。



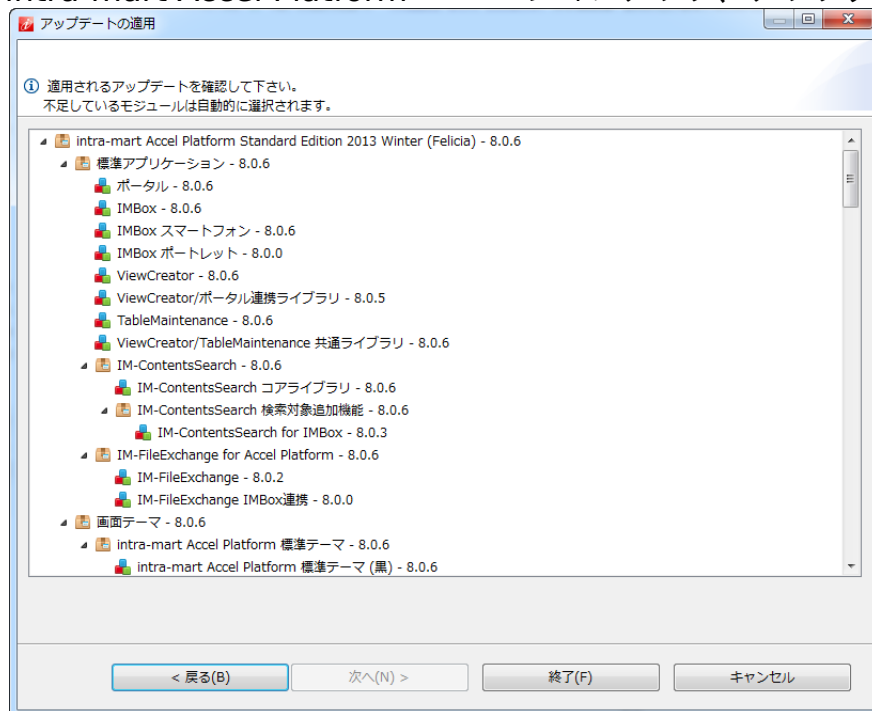
1.3. 「アップデート方法」の選択が表示されますので、選択して「次へ」をクリックしてください。



1.4. 「アップデートを行うバージョン」の選択が表示されますので、最新版を選択して「次へ」をクリックしてください。



1.5. 「適用されるアップデート確認」が表示されますので、問題がなければ、「終了」をクリックすることで、既存のプロジェクトに対してアップデートが実行されます。



2. アップデート終了後、WARファイルを作成し、アプリケーションサーバにデプロイしてください。
3. デプロイ後、アプリケーションサーバを起動し、システム管理画面にて、テナント環境セットアップを実行してください。

intra-mart Accel Platform は、JSSP(JavaScript Server Pages)の実行エンジンとして Rhino を使用しています。Rhino のバージョンを変更したりリリースをまたいでアップデートする場合、一部の JavaScript の挙動が変わります。アップデート前後のバージョンに該当する章を参照してください。

intra-mart Accel Platform 2026 Spring(Mimosa) 以前から 2026 Autumn(Nemesia) 以降へアップデートする場合の Rhino の非互換事項

2026 Autumn(Nemesia) より、JSSP(JavaScript Server Pages)の実行エンジンである Rhino のバージョンを変更します。

これに伴って変わる挙動と、アップデート時に必要な作業は以下の通りです。

概要

intra-mart Accel Platform は、JSSP(JavaScript Server Pages)の実行エンジンとして Rhino を使用しています。IM-LogicDesigner のユーザ定義(Javascript)タスクも同じ Rhino で実行されるため、対象に含まれます。

2026 Autumn(Nemesia) より、Rhino を 1.7R4 から 1.7.15.1 へバージョンアップします。

あわせて JavaScript の言語バージョンを VERSION_1_8 から VERSION_ES6 へ変更します。

これに伴い、一部の JavaScript の挙動が変わります。

2026 Spring(Mimosa) 以前と 2026 Autumn(Nemesia) 以降で挙動が変わる項目を、以降の各章に記載しています。

該当するコードがある場合は、アップデートの前に修正または動作確認を実施してください。

挙動が変わる項目には、BC-001 のような番号を付けています。

BC は Behavior Change の略で、Rhino のバージョンアップで挙動が変わる項目を識別する番号です。

「[影響箇所の確認方法](#)」の linter が出力するレポートも、同じ番号で項目を報告します。

linter で検出できるかどうかは、各項目の「確認方法」に記載しています。

以降の各章の構成は次のとおりです。

1. 影響を受けるコードを洗い出す手順は「[影響箇所の確認方法](#)」に記載しています。
2. 該当するコードの有無にかかわらず計画が必要な作業は「[アップデート時に必要な作業](#)」に記載しています。
3. 挙動が変わる現象の一覧は「[スクリプト開発モデルで互換性が担保できない変更点](#)」以降に記載しています。

影響箇所の確認方法

影響を受けるコードを洗い出すために、JSSP 専用の解析ツール（以下、linter）を提供します。

linter は JavaScript のソースコードを解析し、以降の各章に記載した項目に該当する箇所を報告します。

解析だけを行うツールであり、ソースコードは変更しません。

linter は npm パッケージとして公開しています。

詳細は「[@intra-mart/jssp-rhino-upgrade-17r4to17151-lint](#)」を参照してください。

レポートの項目は BC の番号で報告されます。番号は以降の各章の見出しに付けた BC の番号と同じです。

項目

- [実行方法](#)
- [レポートの見方](#)
- [linter で検出できないもの](#)

実行方法

1. Node.js 22 以上をインストールした端末を用意してください。

2. 解析するディレクトリまたはファイルを指定して、次のコマンドを実行してください。対象は 1 つ以上の指定が必要です。

- Markdown 形式でレポートを出力する場合

```
npx @intra-mart/jssp-rhino-upgrade-17r4to17151-lint --format markdown --lang ja --out report/jssp-lint.md <対象のパス>
```

- Excel 形式でレポートを出力する場合

```
npx @intra-mart/jssp-rhino-upgrade-17r4to17151-lint --format xlsx --lang ja --out report/jssp-lint.xlsx <対象のディレクトリ>
```

3. 出力されたレポートを開き、「[レポートの見方](#)」に従って内容を確認してください。



コラム

- `--lang ja` を省略すると、レポートは英語で出力されます。
- `--format markdown` の場合は `--out` を省略できます。省略した場合は標準出力へ出力します。
- `--format xlsx` の場合は `--out` の指定が必要です。省略した場合はエラーです。
- Markdown 形式と Excel 形式で、レポートの内容は同じです。

レポートの見方

レポートは 5 つのシートで構成されます。Markdown 形式の場合は 5 つの見出しに対応します。

シート	内容
サマリ	対象のパス、生成日時、BC 別の検出件数、ファイル数の内訳を記載します。
検出一覧	コードの修正で対処する項目の検出箇所を、BC 単位で記載します。
移行作業	コードの修正では対処せず、アップデート作業で対処する項目の検出箇所を記載します。
検査対象外	Rhino の拡張構文を含むために解析できなかったファイルを記載します。
構文エラー	解析に失敗したファイルを記載します。

サマリの BC 別一覧表には、検出件数が 0 件の項目も含めてすべての行が出力されます。

「解析したが該当がない」と「そもそも解析していない」を区別できるようにするためです。

「移行作業」のシートには、セッションの破棄と Permanent API のデータ退避に関する検出箇所が出力されます。これらはコードの修正では対処できないため、「[アップデート時に必要な作業](#)」に従って作業を計画してください。該当がない場合も、シートそのものは出力されます。

「検査対象外」のシートに出力されたファイルは、修正が不要という意味ではありません。

Rhino 自身は解析できる記法であるため動作しますが、linter が解析できていないファイルです。以降の各章に記載した項目に該当する記述がないかを、目視で確認してください。

「構文エラー」のシートに出力されたファイルは、解析に失敗したファイルです。

該当するコードを見直してください。

検出箇所には、確認の優先度の目安として影響度を併記します。

影響度は「高」「中」「低」「なし」の 4 段階です。

終了コードは、スクリプトを読み込めなくなる項目を検出したかどうかで決まります。

正規表現のフラグの重複、分割代入のデフォルト値、予約語の 3 項目が対象です。

終了コード 意味

0	スクリプトを読み込めなくなる項目の検出と構文エラーのどちらもありません。
1	スクリプトを読み込めなくなる項目の検出、または構文エラーがあります。
2	コマンドの指定に誤りがあるなど、linter の実行自体に失敗しました。



コラム

- レポートは検出の有無にかかわらず出力されます。可否の判定は終了コードで行ってください。
- サマリの BC 別合計は、検出一覧の総件数と一致しない場合があります。1 つの検出箇所を複数の BC に計上する項目があるためです。
- サマリのファイル数の内訳にある「検出対象外ファイル数」は、製品が標準で提供するファイルとして除外した件数です。

linter で検出できないもの

linter はソースコードの記述だけを解析します。

次の項目は検出できないため、目視での確認が必要です。

1. Rhino の拡張構文（E4X、for each など）を含むファイルの内容です。ファイル単位で検査対象外です。
2. Permanent API と Client API に保存した値そのものです。保存されたデータは解析の対象外です。
3. 変数に代入した Permanent API のインスタンスに対するメソッド呼び出しです。
4. プロパティの列挙順序に依存する処理のように、実行しなければ判定できない項目です。
5. 以降の各章に記載した項目のうち、linter のルールを持たないものです。各項目の「確認方法」を参照してください。



コラム

- Unicode エスケープの記述に誤りがあるコードは、BC の名前ではなく「構文エラー」のシートに出力されます。

アップデート時に必要な作業

本章の作業は、該当するコードの有無にかかわらず、アップデートの計画に組み込んでください。

項目

- [Permanent API の利用](#)
 - [洗い出し](#)
 - [退避](#)
- [Client API の利用](#)
- [互換設定を使うかどうかの判断](#)
- [アップデート手順で必ず行うこと](#)

Permanent API の利用

Permanent API が保存したファイルはストレージに残るため、アップデート後もそのまま読み込まれます。

Rhino 1.7R4 が保存した JavaScript のオブジェクトは Rhino 1.7.15.1 では復元できません。

復元する手段はないため、アップデート前に退避してください。

詳細は「[Client API と Permanent API で保存した JavaScript のオブジェクトが、バージョンアップをまたいで復元できなくなります。\(BC-057\)](#)」を参照してください。

1. linter のレポートの「移行作業」のシートで、Permanent API を利用している箇所を確認してください。
2. linter は `new Permanent(...)` と `Permanent.entries(...)` の記述を検出します。
3. 変数に代入したインスタンスに対するメソッドの呼び出しは検出できません。get や set のほか elements と extinction が該当します。
4. 手順 1 で見つけたスクリプトを目視で確認してください。
5. 保存している値のうち、JavaScript のオブジェクトを保存している箇所を特定してください。配列や Date、関数、独自のプロパティを持つオブジェクトが該当します。

退避

1. アップデート前に、キーごとに get で値を取得してください。
2. 取得した値を JSON.stringify など文字列へ変換し、保存し直してください。
3. アップデート後に文字列から復元するよう、スクリプトを修正してください。



コラム

- 文字列だけを保存している場合は、退避は不要です。
- 関数や getter、setter を含む値は文字列へ変換できないため、この手順の対象外です。
- カテゴリ内の全エントリを読む elements と extinction は、キーごとに値を復元します。復元できないキーの値だけが null になり、呼び出し自体は成功します。文字列を保存したキーは、そのまま読めます。
- カテゴリ内のキー名の一覧を返す entries は、値の復元を伴わないため影響を受けません。

Client API の利用

Client API が保存した値はセッションに保持されます。

Rhino 1.7R4 のセッションが残った状態で Rhino 1.7.15.1 を起動すると、保存した JavaScript のオブジェクトを取り出せません。

取り出しに失敗した場合は、警告ログを出力して null を返します。例外にはなりません。

値を取り出せないことに気付きにくいので、注意してください。

1. セッション管理モジュール や セッション管理 組込Hazelcast連携 を利用しているかを確認してください。
2. 利用している場合は、アップデート作業の中でセッションを破棄する手順を計画してください。
3. アップデート後は再ログインが必要である旨を、利用者へ案内する準備をしてください。



コラム

- アプリケーションサーバ自身がセッションを永続化する設定の場合も、破棄の対象です。停止時にセッションをファイルへ書き出し、起動時に読み戻す設定が該当します。
- Client API の destroy と removeSession は、実行中のリクエストのセッションを対象とする内部向けの機能です。本章のセッションの破棄とは別のものです。

互換設定を使うかどうかの判断

Java の API が返す数値の受け渡し方式が変わります。既定は 2026 Autumn(Nemesia) 以降の挙動ですが、設定ファイルで 2026 Spring(Mimosa) 以前の挙動へ戻せます。

設定の内容は「[Java の API が返す BigDecimal などの数値の受け渡し方式が変わります。\(BC-045\)](#)」を参照してください。

1. Java の API から BigDecimal や BigInteger を受け取って計算しているスクリプトがあるかを確認してください。
2. ある場合は、スクリプトを 2026 Autumn(Nemesia) 以降の挙動にあわせて修正するか、互換設定で 2026 Spring(Mimosa) 以前の挙動へ戻すかを決めてください。

3. 互換設定を使う場合は、アップデート作業の中で設定ファイルを配置する手順を計画してください。

アップデート手順で必ず行うこと

1. アップデート前に、Permanent API に保存した JavaScript のオブジェクトの退避が完了していることを確認してください。
2. サーバを停止し、保持されているセッションをすべて破棄してください。セッション管理モジュールのほか、アプリケーションサーバ自身による永続化も対象です。
3. IM-Juggling で war ファイルを作成し、アップデートを実施してください。
4. 互換設定を使用する場合は、システムストレージの conf ディレクトリに jssp-compatibility-config.xml を配置してください。
5. サーバを起動してください。互換設定は起動時に一度だけ読み込まれます。設定を変更した場合も再起動が必要です。
6. 利用者に再ログインを案内してください。



注意

- セッションを破棄しないまま Rhino 1.7.15.1 を起動すると、Client API で保存した JavaScript のオブジェクトの取得が、警告ログとともに null を返します。
- セッションの破棄は、新しいバージョンがリクエストを受け付ける前に完了させてください。

スクリプト開発モデル で互換性が担保できない変更点

本章は、2026 Spring(Mimosa) 以前と 2026 Autumn(Nemesia) 以降で挙動が変わる現象の一覧です。各項目には、linter で検出できるかどうかを「確認方法」として併記しています。

項目

- 数値・文字列・オブジェクト
 - JSON.parse が、規格に従わない JSON を受け付けなくなります。(BC-001)
 - Number と + 演算子による文字列から数値への変換結果が変わります。(BC-010)
 - parseInt で基数を省略した場合に、先頭が 0 の文字列を 8 進数として解釈しなくなります。(BC-011)
 - Array の sort で、空要素が空要素のまま末尾へ寄ります。(BC-033)
 - this が null または undefined の状態で組み込みメソッドを呼び出すと例外が発生します。(BC-034)
 - Object.prototype.toString を null と undefined に適用した場合の戻り値が変わります。(BC-035)
 - String の split に undefined を渡しても、文字列 “undefined” では分割しなくなります。(BC-036)
- 日付
 - ISO 8601 形式の文字列を Date に渡しても Invalid Date にならなくなります。(BC-005)
 - タイムゾーンを指定しない日時の文字列が、UTC ではなくローカル時刻として解釈されます。(BC-005)
- 正規表現
 - 正規表現の {0} が、直前の文字に誤ってマッチしなくなります。(BC-016)
 - 正規表現のフラグを重複して指定するとスクリプトを読み込めなくなります。(BC-018)
- 構文・予約語
 - 分割代入の宣言にデフォルト値を書くとスクリプトを読み込めなくなります。(BC-023)
 - class、import などの予約語を変数名・関数名・引数名に使えなくなります。(BC-025)
- strict mode
 - “use strict” を指定したコードで、読み取り専用プロパティへの代入と構成不可プロパティの delete で TypeError が発生します。(BC-029)
- Java 連携
 - Java の配列を Array の concat に渡すと、要素が展開されます。(BC-032)
 - Java の List を for...in で列挙した結果が、メソッド名からインデックスに変わります。(BC-042)
 - Java の Number をラップしたオブジェクトを + 演算子で文字列化した結果が、JavaScript の数値表現に変わります。(BC-045)
 - Java の API が返す BigDecimal などの数値の受け渡し方式が変わります。(BC-045)
- エラーオブジェクト
 - ConversionError が定義されなくなります。(BC-039)
- 組み込みメソッドの追加
 - String、Array、Number、Object に組み込みメソッドが追加されます。(BC-041)
 - ES6 のグローバルオブジェクトが追加されます。(BC-041)
- ジェネレータ
 - ジェネレータの next が、yield した値ではなくオブジェクトを返します。(BC-052)
- セッション、Permanent の保存データ
 - Client API と Permanent API で保存した JavaScript のオブジェクトが、バージョンアップをまたいで復元できなくなります。(BC-057)

数値・文字列・オブジェクト

JSON.parse が、規格に従わない JSON を受け付けなくなります。(BC-001)

- 2026 Spring(Mimosa) 以前: 末尾のカンマや小数点で終わる数値を含む JSON も、エラーにならず解析できていました。
- 2026 Autumn(Nemesia) 以降: 規格に従わない JSON を渡すと SyntaxError が発生します。

```
JSON.parse('[1,]');
// 変更前: [1]
// 変更後: SyntaxError

JSON.parse('{ "a":10.}');
// 変更前: { "a":10}
// 変更後: SyntaxError
```

- 影響を受けるコード: 外部システムやデータベースから受け取った JSON を JSON.parse で解析しているスクリプトです。
文字列を組み立てて JSON.parse に渡している箇所も、末尾のカンマが残ると失敗します。
シングルクォートのキーや引用符のないキーは、2026 Spring(Mimosa) 以前でもエラーになっていたため変わりません。
- 対処: JSON.parse に渡す文字列が規格に従っているかを確認してください。
- 確認方法: linter では検出できません。目視で確認してください。

Number と + 演算子による文字列から数値への変換結果が変わります。(BC-010)

- 2026 Spring(Mimosa) 以前: `Number("-0x10")` は -16 を返し、`Number("0b101")` は NaN を返していました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、符号付きの 16 進表記は NaN を返し、2 進表記と 8 進表記は数値を返します。

```
Number("-0x10");
// 変更前: -16
// 変更後: NaN

Number("0b101");
// 変更前: NaN
// 変更後: 5
```

- 影響を受けるコード: 文字列を Number や単項の + 演算子で数値へ変換しているスクリプトです。
算術演算や == による比較での暗黙の変換も同じ結果です。
- 対処: 数値へ変換する文字列の形式を確認してください。16 進の文字列は parseInt に基数 16 を指定して変換してください。
- 確認方法: linter で検出できます。

parseInt で基数を省略した場合に、先頭が 0 の文字列を 8 進数として解釈しなくなります。(BC-011)

- 2026 Spring(Mimosa) 以前: 先頭が “0” の文字列を無条件に 8 進数として解釈していました。
- 2026 Autumn(Nemesia) 以降: 10 進数として解釈します。

```
parseInt("010");
// 変更前: 8
// 変更後: 10

parseInt("08");
// 変更前: NaN
// 変更後: 8
```

- 影響を受けるコード: 基数を省略した parseInt に、日付や時刻の “08”、“09” のようなゼロ埋めの文字列を渡しているスクリプトです。
多くの場合は意図しない結果が正しくなる方向の変更ですが、NaN になることを前提にした分岐は成立しなくなります。

- 対処: parseInt の第 2 引数に基数 10 を明示してください。
- 確認方法: linter で検出できます。

Array の sort で、空要素が空要素のまま末尾へ寄ります。(BC-033)

- 2026 Spring(Mimosa) 以前: sort の際に、空要素を値が undefined の実プロパティへ変換していました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、空要素を undefined よりさらに後ろへ寄せ、空要素のまま保ちます。

```
var a = ["b", undefined, "a", , "c"];
a.sort();
"4" in a;
// 変更前: true
// 変更後: false
```

- 影響を受けるコード: sort した後の配列に対して、in 演算子、hasOwnProperty、for...in でプロパティの有無を判定しているスクリプトです。
要素の値と個数は変わりません。
- 対処: sort した後の配列をプロパティの有無で判定している箇所を確認してください。
- 確認方法: linter では検出できません。目視で確認してください。



コラム

矛盾した比較関数を sort に渡した場合、要素数が 32 以上で矛盾が検出されたときに `IllegalArgumentException` が発生します。

この例外は JSP の try / catch では捕捉できず、リクエストはエラーで終了します。
検出されるかどうかは入力に依存します。

この動作は 2026 Spring(Mimosa) 以前でも同様であり、バージョンアップによる変更ではありません。

this が null または undefined の状態で組み込みメソッドを呼び出すと例外が発生します。(BC-034)

- 2026 Spring(Mimosa) 以前: `Array.prototype.slice.call(null)` のような呼び出しは、グローバルオブジェクトを対象として動作していました。
動作するもの、文字列として処理するもの、Java の例外になるものが混在していました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、一貫して `TypeError` が発生します。

```
Array.prototype.slice.call(null);
// 変更前: 例外にならない
// 変更後: TypeError
```

- 影響を受けるコード: call や apply の第 1 引数に、null または undefined になり得る変数を渡しているスクリプトです。
配列風のオブジェクトを配列へ変換する共通関数が該当しやすい箇所です。
- 対処: call や apply の第 1 引数が null または undefined にならないことを確認してください。
- 確認方法: linter では検出できません。目視で確認してください。

Object.prototype.toString を null と undefined に適用した場合の戻り値が変わります。(BC-035)

- 2026 Spring(Mimosa) 以前: null と undefined のどちらも `[object Object]` を返していました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、それぞれ `[object Null]`、`[object Undefined]` を返します。

```
Object.prototype.toString.call(null).slice(8, -1);
// 変更前: "Object"
// 変更後: "Null"
```

- 影響を受けるコード: `Object.prototype.toString.call(x).slice(8, -1)` の形で値の型名を取得しているユーティリティ関数です。
null と undefined を渡したときの戻り値が変わるため、その戻り値で分岐している処理の結果が変わります。
- 対処: 型名を取得するユーティリティ関数と、その戻り値で分岐している箇所を確認してください。
- 確認方法: linter で検出できます。

String の split に undefined を渡しても、文字列 “undefined” では分割しなくなります。(BC-036)

- 2026 Spring(Mimosa) 以前: 区切り文字の undefined を文字列 “undefined” に変換して分割していました。
- 2026 Autumn(Nemesia) 以降: 分割せず、文字列全体を 1 要素の配列として返します。

```
'fooundefinedbar'.split(undefined);
// 変更前: ['foo', 'bar']
// 変更後: ['fooundefinedbar']
```

- 影響を受けるコード: 変数が undefined のまま split に渡し、かつ対象の文字列に “undefined” が含まれる場合に限られます。
- 対処: split に渡す区切り文字が undefined にならないことを確認してください。
- 確認方法: linter では検出できません。目視で確認してください。

日付

ISO 8601 形式の文字列を Date に渡しても Invalid Date にならなくなります。(BC-005)

- 2026 Spring(Mimosa) 以前: ISO 8601 形式の文字列を渡した Date は Invalid Date になっていました。
`new Date("2024-01-01")` や `new Date("2024-01-01T00:00:00Z")` が該当します。
`getTime()` は NaN を返していました。
その Date に `toISOString()` を呼ぶと `RangeError` になりました。
ただし `2024-01-01T00:00:00.000Z` のような 24 文字固定の形式だけは、2026 Spring(Mimosa) 以前でもパースできていました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおりパースされます。24 文字固定の形式の結果は変わりません。

```
new Date("2024-01-01").getTime();
// 変更前: NaN
// 変更後: 正しい時刻の数値
```

- 影響を受けるコード: 文字列を Date に渡し、Invalid Date になることを利用して入力の妥当性を判定しているスクリプトです。
数値を引数にして Date を生成しているコードは影響を受けません。
- 対処: Invalid Date の判定による書式チェックを、正規表現などによる明示的な書式チェックへ置き換えてください。
- 確認方法: linter で検出できます。

タイムゾーンを指定しない日時の文字列が、UTC ではなくローカル時刻として解釈されます。(BC-005)

- 2026 Spring(Mimosa) 以前: タイムゾーンのない日時の文字列は Invalid Date になっていました。
`new Date("2024-05-03T10:00:00")` のような文字列が該当します。

- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、サーバのタイムゾーンによるローカル時刻として解釈されます。
日本標準時のサーバでは、UTC として解釈した場合と 9 時間の差が生じます。
日付だけの "2024-05-03" は、2026 Spring(Mimosa) 以前と同じく UTC として解釈されます。タイムゾーンを指定した文字列の結果も変わりません。

```
new Date("2024-05-03T10:00:00").getTime();
// 変更前: NaN
// 変更後: ローカル時刻の 2024-05-03 10:00 を表す数値
```

- 影響を受けるコード: タイムゾーンを含まない日時の文字列を Date に渡しているスクリプトです。
例外にならず値だけが変わります。気付かないまま時刻がずれ、画面表示やデータベースの登録内容に影響します。
- 対処: 日時の文字列にタイムゾーンを明示するか、年月日と時分秒を数値で指定して Date を生成してください。
- 確認方法: linter で検出できます。

正規表現

正規表現の {0} が、直前の文字に誤ってマッチしなくなります。(BC-016)

- 2026 Spring(Mimosa) 以前: a{0} は「0 回の繰り返し」ではなく、直前の文字にマッチしていました。
- 2026 Autumn(Nemesia) 以降: ECMAScript の規定どおり、0 回の繰り返しとして空文字にマッチします。

```
"aaa".match(/a{0}/)[0];
// 変更前: "aaa"
// 変更後: ""

"aaa".replace(/a{0}/g, "-");
// 変更前: "--"
// 変更後: "-a-a-a-"
```

- 影響を受けるコード: {0} を含む正規表現でマッチや置換をしているスクリプトです。
繰り返し回数を変数から組み立てていて、値が 0 になり得る箇所も該当します。
- 対処: {0} を含む正規表現を見直し、意図した結果になるかを確認してください。
- 確認方法: linter で検出できます。

正規表現のフラグを重複して指定するとスクリプトを読み込めなくなります。(BC-018)

- 2026 Spring(Mimosa) 以前: /x/gg のようにフラグが重複していても、エラーにならず読み込んでいました。
- 2026 Autumn(Nemesia) 以降: SyntaxError が発生します。

```
new RegExp("a", "gg");
// 変更前: 例外にならない
// 変更後: SyntaxError
```

- 影響を受けるコード: 同じフラグを 2 つ以上指定している正規表現です。
正規表現リテラルに記述している場合は、該当箇所だけでなくスクリプト全体が読み込めなくなります。
- 対処: 重複しているフラグを取り除いてください。
- 確認方法: linter で検出できます。



コラム

未サポートのフラグを指定した場合のエラーメッセージに、対象のフラグ文字が含まれます。
エラーメッセージの文字列で分岐している箇所だけが確認の対象です。

構文・予約語

分割代入の宣言にデフォルト値を書くとスクリプトを読み込めなくなります。(BC-023)

- 2026 Spring(Mimosa) 以前: `var [a = 10] = b;` のような記述は読み込めていました。
ただし実行するとエンジン内部の例外となり、JSP の try / catch では捕捉できずリクエストはエラーで終了していました。
- 2026 Autumn(Nemesia) 以降: スクリプトの読み込み時に `SyntaxError` が発生します。
エラーを検出する時点が、実行時から読み込み時へ移ります。

```
var [a = 10] = [];
// 変更前: 実行時に捕捉できない例外
// 変更後: 読み込み時に SyntaxError
```

- 影響を受けるコード: 分割代入の宣言にデフォルト値を記述しているスクリプトです。
該当箇所だけでなく、スクリプト全体が読み込めなくなります。
- 対処: デフォルト値の記述をやめ、分割代入の後に値を判定して既定値を代入してください。
- 確認方法: linter で検出できます。

class、import などの予約語を変数名・関数名・引数名に使えなくなります。(BC-025)

- 2026 Spring(Mimosa) 以前: class、enum、export、extends、import、super、await を識別子として使っても、スクリプトを読み込めていました。
- 2026 Autumn(Nemesia) 以降: これらを識別子として使うと、スクリプトの読み込み時に `SyntaxError` が発生します。
JavaScript の言語バージョンが `VERSION_ES6` になったためです。
プロパティ名として使う場合 (`o.class` や `{class: 1}`) は、引き続き使えます。
- 影響を受けるコード: 上記の語を変数名・関数名・引数名に使っているスクリプトです。該当箇所だけでなく、スクリプト全体が読み込めなくなります。
- 対処: 該当する識別子を別の名前に変更してください。
- 確認方法: linter で検出できます。

strict mode

“use strict” を指定したコードで、読み取り専用プロパティへの代入と構成不可プロパティの `delete` で `TypeError` が発生します。(BC-029)

- 2026 Spring(Mimosa) 以前: “use strict” の効果はパース時のチェックだけで、実行時の強制はありませんでした。
読み取り専用プロパティへの代入も、構成不可プロパティの `delete` も、例外にならず無視されていました。
- 2026 Autumn(Nemesia) 以降: どちらも実行時に `TypeError` が発生します。
- 影響を受けるコード: “use strict” を記述したサーバサイドのスクリプトで、2026 Spring(Mimosa) 以前では無視されていた代入や `delete` を行っている箇所です。
- 対処: “use strict” を記述したスクリプトで、読み取り専用プロパティへの代入と構成不可プロパティの `delete` を行っていないか確認してください。
- 確認方法: linter で検出できます。



コラム

eval に渡した “use strict” のコードで未宣言の変数へ代入した場合は、2026 Spring(Mimosa) 以前と同じく ReferenceError が発生します。

Java 連携

Java の配列を Array の concat に渡すと、要素が展開されます。(BC-032)

- 2026 Spring(Mimosa) 以前: Java の配列は展開されず、1 つの要素として連結されていました。
- 2026 Autumn(Nemesia) 以降: Java の配列の要素が展開されて連結されます。
- 影響を受けるコード: Java の API が返す配列を、JavaScript の配列の concat に渡しているスクリプトです。連結後の配列の長さや要素の並びが変わります。
- 対処: Java の配列を 1 つの要素として連結したい場合は、push で追加してください。
- 確認方法: linter では検出できません。目視で確認してください。



コラム

Array.prototype を継承した JavaScript のオブジェクトを concat に渡した場合の結果は変わりません。

Java の List を for...in で列挙した結果が、メソッド名からインデックスに変わります。(BC-042)

- 2026 Spring(Mimosa) 以前: java.util.List を for...in で列挙すると、List が持つメソッドの名前が返っていました。
また list[0] のようなインデックスでのアクセスは例外になっていました。
- 2026 Autumn(Nemesia) 以降: for...in は要素のインデックスを返します。インデックスでの読み書きもできます。

```
var list = new java.util.ArrayList();
list.add("x");
var keys = [];
for (var k in list) { keys.push(k); }
keys.length;
// 変更前: 37 (メソッド名の個数)
// 変更後: 1 (インデックスの個数)
```

- 影響を受けるコード: Java の API が返す List を for...in で列挙しているスクリプトです。
インデックスでのアクセスが例外になることを前提にした try / catch も、例外が発生しなくなります。
- 対処: List の列挙には、size と get を使った添字による繰り返しを使用してください。
- 確認方法: linter では検出できません。目視で確認してください。



コラム

java.util.Map をプロパティ名で読み書きする挙動は、初期状態では無効です。
Map に対する記述は、2026 Spring(Mimosa) 以前と同じく Bean のプロパティとメソッドだけが対象です。

Java の Number をラップしたオブジェクトを + 演算子で文字列化した結果が、JavaScript の数値表現に変わります。(BC-045)

- 2026 Spring(Mimosa) 以前: java.lang.Double をラップしたオブジェクトを + 演算子で文字列化すると “1.0” になっていました。
Java の Double#toString の表現です。

- 2026 Autumn(Nemesia) 以降: JavaScript 標準の数値表現である “1” です。
- 影響を受けるコード: JSP の既定の設定では、Java のメソッドの戻り値の Number はラップされずにそのまま渡するため、この経路の結果は変わりません。
2026 Spring(Mimosa) 以前と 2026 Autumn(Nemesia) 以降のどちらも “1” です。
結果が変わるのは、スクリプトの中で `new java.lang.Double(...)` のように Java の Number オブジェクトを明示的に生成し、+ 演算子で連結している場合です。
String(obj) による明示的な文字列変換は、2026 Spring(Mimosa) 以前と 2026 Autumn(Nemesia) 以降のどちらも “1.0” のままです。
- 対処: Java の Number オブジェクトを明示的に生成して文字列連結している箇所を確認してください。
- 確認方法: linter では検出できません。目視で確認してください。

Java の API が返す BigDecimal などの数値の受け渡し方式が変わります。(BC-045)

- 2026 Spring(Mimosa) 以前: Java の API が返す BigDecimal などは JavaScript の number へ変換されていました。変換の際に精度が失われることがありました。
- 2026 Autumn(Nemesia) 以降: BigDecimal などは Java のオブジェクトをラップしたまま渡ります。BigInteger は JavaScript の BigInt に変換されます。
- 影響を受けるコード: Java の API の戻り値の数値を、そのまま算術演算や比較に使用しているスクリプトです。
- 対処: 設定ファイルで 2026 Spring(Mimosa) 以前の挙動へ戻せます。
システムストレージの `conf/jssp-compatibility-config.xml` に `legacy-java-number-wrap` 要素を配置してください。
要素の値には true を指定してください。
要素を省略した場合と設定ファイルを配置しない場合は、2026 Autumn(Nemesia) 以降の挙動です。
設定は起動時に一度だけ反映されるため、変更した後はサーバを再起動してください。
- 確認方法: linter で検出できます。



注意

この設定を有効にしても、BigInteger を Java 側へ渡し直したときに同一のインスタンスが保たれる 2026 Spring(Mimosa) 以前の挙動は再現されません。
ラップする際に double へ変換されるためです。
BigDecimal は同一のインスタンスのまま往復します。



コラム

java.util.Map を引数にとるメソッドと、独自のインタフェースを引数にとるメソッドがオーバーロードされている場合の動作も変わります。
JavaScript のオブジェクトリテラルまたは関数を渡したときが対象です。
2026 Spring(Mimosa) 以前では呼び出すメソッドを一意に決定できずエラーになっていましたが、java.util.Map を引数にとるメソッドが選択されて成功します。
エラーになることを前提にした catch ブロックでの分岐だけが確認の対象です。

エラーオブジェクト

ConversionError が定義されなくなります。(BC-039)

- 2026 Spring(Mimosa) 以前: 標準にはないエラーコンストラクタ ConversionError がグローバルに定義されていました。
- 2026 Autumn(Nemesia) 以降: ConversionError は定義されず、参照すると ReferenceError が発生します。
- 影響を受けるコード: ConversionError を throw している、または catch ブロックで instanceof ConversionError の判定を行っているスクリプトです。

- 対処: 標準の `TypeError` や `Error` に置き換えてください。
- 確認方法: `linter` では検出できません。目視で確認してください。

組み込みメソッドの追加

String、Array、Number、Object に組み込みメソッドが追加されます。(BC-041)

- 2026 Spring(Mimosa) 以前: 以下のメソッドは存在せず、`typeof` は `undefined` でした。
- 2026 Autumn(Nemesia) 以降: 以下のメソッドが利用できます。
 - ・ `String.prototype` の `includes`、`startsWith`、`endsWith`、`repeat`、`normalize`、`codePointAt`、`padStart`、`padEnd`、`at`、`trimStart`、`trimEnd`、`replaceAll` など
 - ・ `Array.prototype` の `find`、`findIndex`、`fill`、`includes`、`copyWithin` と、`Array.from`、`Array.of`
 - ・ `Number` の `isInteger`、`isFinite`、`isNaN`、`isSafeInteger`
 - ・ `Object` の `assign`、`is`
- 影響を受けるコード: 上記と同名のメソッドを `prototype` に独自に追加している、またはポリフィルとして定義しているスクリプトです。
`X.prototype.foo = X.prototype.foo || ...` のように存在チェックを伴うポリフィルは、組み込みの実装に置き換わります。
 無条件に上書きするポリフィルは、2026 Spring(Mimosa) 以前と同じく独自の実装が使われます。
- 対処: ポリフィルを定義している箇所を確認してください。独自の実装と組み込みの実装で引数の扱いなどに差がある場合、結果が変わります。
- 確認方法: `linter` で検出できます。



コラム

正規表現の `sticky` フラグ `y` も利用できます。
 機能の追加であるため、`y` フラグが `SyntaxError` になることを前提にした判定だけが確認の対象です。

ES6 のグローバルオブジェクトが追加されます。(BC-041)

- 2026 Spring(Mimosa) 以前: 以下のグローバルオブジェクトは存在しませんでした。
- 2026 Autumn(Nemesia) 以降: 以下のグローバルオブジェクトが利用できます。
 - ・ `Symbol`、`Map`、`Set`、`WeakMap`、`WeakSet`、`Promise`
 - ・ `typed array` 系のコンストラクタ
 - ・ `Object.entries`、`Object.values`、`Object.setPrototypeOf`
 - ・ `globalThis`
- 影響を受けるコード: 上記と同名のグローバル変数を定義している、またはポリフィルとして定義しているスクリプトです。
- 対処: 同名のグローバル変数を定義している箇所を確認してください。
- 確認方法: `linter` で検出できます。

ジェネレータ

ジェネレータの `next` が、`yield` した値ではなくオブジェクトを返します。(BC-052)

- 2026 Spring(Mimosa) 以前: `function f() { yield 1; }` のようなジェネレータの `next` は、`yield` した値そのものを返していました。
- 2026 Autumn(Nemesia) 以降: JavaScript の言語バージョンが `VERSION_ES6` になったことにより、`value` と `done` を持つオブジェクトを返します。

```
function f() { yield 1; }
var g = f();
g.next();
// 変更前: 1
// 変更後: { value: 1, done: false }
```

- 影響を受けるコード: yield を使った関数の戻り値に対して next を呼んでいるスクリプトです。
- 対処: next の戻り値の value プロパティを参照し、終了の判定は done プロパティで行ってください。
- 確認方法: linter では検出できません。目視で確認してください。

セッション、Permanent の保存データ

Client API と Permanent API で保存した JavaScript のオブジェクトが、バージョンアップをまたいで復元できなくなります。(BC-057)

- 2026 Spring(Mimosa) 以前: Rhino 1.7R4 が保存した JavaScript の値は、同じ Rhino 1.7R4 で復元できていました。
オブジェクト、配列、Date、String オブジェクト、関数が該当します。
- 2026 Autumn(Nemesia) 以降: Rhino 1.7.15.1 は、Rhino 1.7R4 が保存したこれらの値を復元できません。
保存形式がバージョン間で変わっており、古い形式を読み替える仕組みは存在しないため、復元する手段はありません。
文字列として保存した値は、バージョンをまたいで復元できます。
- 影響を受けるコード:
 - ・ Client API では Client.get が値の復元に失敗します。失敗時は警告ログを出力し、null を返します。keys、remove、set は値の復元を伴わないため動作します。
 - ・ セッション管理モジュール などを利用し、バージョンアップをまたいでセッションが残る構成で顕在化します。
 - ・ Permanent API では、保存先のファイルがバージョンの差し替えで消えないため、オブジェクトを保存した値の復元を伴う get、remove、set が警告ログとともに null を返します。
 - ・ Permanent API のカテゴリ内の全エントリを読む elements、extinction は、復元できないキーの値だけを null として返します。文字列を保存したキーの値は読めます。
 - ・ JSP の duplicate() は同一の JVM の中で保存と復元をするためバージョンをまたがず、影響を受けません。
- 対処: 「[アップデート時に必要な作業](#)」の Permanent API の退避とセッションの破棄を実施してください。
- 確認方法: linter で検出できます。レポートの「移行作業」のシートに出力されます。



注意

復元する手段はないため、アップデート前の退避とセッションの破棄を省略すると、保存した値は失われます。



コラム

復元に失敗した場合は、例外ではなく、警告ログを出力して null を返します。
呼び出し自体は成功するため、値を取り出せないことに気付きにくい点に注意してください。

IM-LogicDesigner のユーザ定義(JavaScript)タスクで互換性が担保できない変更点

項目

- JSSP と共通の変更点
- ユーザ定義(JavaScript)タスク固有の変更点
 - ユーザ定義(JavaScript)タスクから 6 つの名前が利用できなくなります。

JSSP と共通の変更点

ユーザ定義(JavaScript)タスクは、JSSP と同じ Rhino、同じ JavaScript の言語バージョンで実行されます。このため、「[スクリプト開発モデル で互換性が担保できない変更点](#)」に記載した変更点は、すべてユーザ定義(JavaScript)タスクにも当てはまります。

同章の各項目を確認してください。

linter はディレクトリまたはファイルを解析の対象とします。

タスクの JavaScript を linter にかける場合は、スクリプトをファイルとして保存し、対象に指定してください。

ユーザ定義(JavaScript)タスク固有の変更点

ユーザ定義(JavaScript)タスクから 6 つの名前が利用できなくなります。

- 2026 Spring(Mimosa) 以前: 以下の 6 つの名前が利用できていました。
- 2026 Autumn(Nemesia) 以降: ユーザ定義(JavaScript)タスクには、危険な名前をスクリプトから見えなくする一覧が適用されます。以下の 6 つの名前がこの一覧に追加され、利用できなくなります。
 - ・ defineClass: クラスパス上のクラスを名前を読み込み、JavaScript から使えるようにします。
 - ・ deserialize: ファイルを読み、Java のデシリアライズを行います。
 - ・ environment: JVM のシステムプロパティを読み書きします。
 - ・ getClass: Java オブジェクトから java.lang.Class を取得します。
 - ・ JavaAdapter: JavaScript で Java のインタフェースの実装や継承を作ります。
 - ・ Continuation: インタプリタの継続を捕捉します。

参照すると undefined を返し、関数として呼び出すと TypeError が発生します。

- 影響を受けるコード: 上記の名前を利用しているユーザ定義(JavaScript)タスクです。
 - 対処: これらの名前の利用をやめてください。Java との連携が必要な処理は、Java で実装したタスクへ移してください。
- 一覧は運用者が編集できますが、いずれも危険な操作の入口であるため、一覧から外すことは推奨しません。
- 確認方法: linter では検出できません。目視で確認してください。



コラム

2026 Autumn(Nemesia) 以降で追加された globalThis は、ユーザ定義(JavaScript)タスクではスクリプト自身のグローバルを指すように束縛されています。

`globalThis === this` は true です。

書き換えと削除はできません。

2026 Spring(Mimosa) 以前には存在しなかった名前であるため、既存のユーザ定義(JavaScript)タスクは影響を受けません。

Java API（モジュール開発者向け）

本章は、独自に開発したモジュールから Rhino の Java API を使用している場合の変更点です。

linter は JavaScript だけを解析するため、本章の項目は検出できません。

項目

- 一部の内部クラスが提供されなくなります。
- 一部の Java API のシグネチャと非推奨の扱いが変わります。

一部の内部クラスが提供されなくなります。

- 2026 Spring(Mimosa) 以前: `v8dtoa` パッケージの旧実装や、jdk13、jdk15 向けの `VMBridge` などの内部クラスが提供されていました。
- 2026 Autumn(Nemesia) 以降: これらの内部クラスは提供されません。
- 影響を受けるコード: 独自に開発したモジュールから、上記の内部クラスを参照している場合です。参照している箇所は、再コンパイルをしていなくても、実行時のクラス読み込みで `NoClassDefFoundError` が発生します。
- 対処: 独自に開発したモジュールから、提供されなくなった内部クラスへの参照を除去してください。

一部の Java API のシグネチャと非推奨の扱いが変わります。

- 2026 Spring(Mimosa) 以前: `ContextAction` は型引数を持ちませんでした。 `Context.enter()` と `Context` のコンストラクタは非推奨ではありませんでした。
- 2026 Autumn(Nemesia) 以降: `ContextAction` が `ContextAction<T>` へジェネリクス化されます。型引数を指定しない利用は警告のみでコンパイルできます。
`Context.enter()` と `Context` のコンストラクタは非推奨です。動作は変わりません。
- 影響を受けるコード: 独自に開発したモジュールのうち、上記の API を使用している箇所です。
- 対処: コンパイル時の警告を解消する場合は、`ContextAction` に型引数を指定してください。



コラム

JSR-223 の `ScriptEngine` は提供されません。