# CreateFlowRequestflow\_requestフローを利用

# **1. 機能の概要:**

フローテンプレートを選択して申請フローを起票し、申請を行う画面です。

# **2. 目的:**

承認の過程で必要な要素（承認位置, 承認者, 添付ファイル, 選択されたユーザーやグループ）を効率的に選択し, 適切な承認処理を進行できるようにします。

# **3. 画面構想（後で記載）**

１．新規申請

２．フロー状態一覧

３．保存、申請、中止、削除

４．承認

５．引き上げ

６．引き下げ

７．差し戻し

８．否認

９．一括承認

10.　フロー参照（進捗）

# **4. データオブジェクト一覧 ・ビジネスロジック**

## **①機能：新規申請**

関連クラスとアクション：

モデルクラス：FlowRequestData

アクション：CreateFlowRequest

説明：[新規申請]ボタンをクリックして申請フロー選択画面に遷移します。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | パラメータ |  | ダミーデータ | 説明 |
|  | flow\_request\_id | 申請フローID | SKT001… |  |
|  | flow\_request\_name | 申請フロー名/概要 | 稟議フロー |  |
|  | flow\_id | フローID | F1 |  |
|  | form\_id | 帳票ID | T1 |  |
|  | form\_name | 帳票名 | 稟議書 |  |
|  | node\_id | 承認者位置 | R1 |  |
|  | user\_id | 申請フロー作成者ID | 32430… |  |
|  | user\_name | 申請フロー作成者名 | クリエイト ユーザ03 |  |
|  | user\_group\_id | ユーザーグループId | 2… |  |
|  | user\_group\_name | ユーザーグループ名 | IT人材 |  |
|  | user\_position | 役職 | 一般 |  |
|  | process\_function | 行った機能 | 申請する | 保存、申請、中止、削除・・・ |

## **ビジネスロジック：新規申請**

関連クラスとアクション：

モデルクラス：FlowRequestData

アクション：CreateFlowRequest

説明：[新規申請]ボタンをクリックして申請フロー選択画面に遷移します。［フロー一覧］から、起票するフローテンプレートを選択します。申請フローID・申請フロー名が変更できる場合や、申請フロー作成グループが選択できる場合は、 申請フロー情報指定ダイアログが表示されますので、各項目を入力して［申請する］ボタンをクリ ックします。自動的に次の帳票画面または申請指示画面に進みます。

**申請フローID・申請フロー名はフローテンプレート作成の設計書の　パラメータ説明の採番ルールと命名規則の通り設定できます。**

フォームから受け取ったデータを格納するオブジェクト作成

クラス:FlowRequest

アクション: CreateFlowRequest（FlowRequestData inputData）

// 入力データを整形し、必要な形式に変換

        var flowRequestData= new FlowRequestData

        {

flow\_request\_id=inputdata.flow\_request\_id

flow\_request\_name=inputdata.flow\_request\_name

flow\_id=inputdata.flow\_id

flow\_name=inputdata.flow\_name

form\_id=inputdata.form\_id

form\_name= inputdata.form\_name

user\_id=inputdata.user\_id

user\_name=inputdata.user\_name

user\_group\_id=inputdata.group\_id

user\_position=inputdata.user\_position

process\_function=inputdata.process\_function

        };

戻り値：flowRequestData

flowRequestDataをflow\_requestテーブルにデータを挿入します。

DBに接続処理

データベース操作を行っています。

挿入query

**※同時に申請フロー状態もflow\_approval\_resultテーブル挿入にします。**

## **②機能：フロー状態一覧**

関連クラスとアクション：

モデルクラス：FlowApprovalResultData

アクション：FlowApprovalResult

説明：申請フローの中止、中止取消、削除や処理済の承認位置へなどのフロー状態のメンテナンスを行います。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | パラメータ |  | ダミーデータ | 説明 |
|  | flow\_request\_id | 申請フローID | SKT001… |  |
|  | node\_id | ノードID | R1 |  |
|  | approve\_type | 承認者タイプ | 申請フロー作成者 | グループ・ユーザー |
|  | require\_position | 役職 | 一般 |  |
|  | approve\_id | 承認者ID | 324301.. |  |
|  | status | 状態 | 申請する・申請待ち | 申請者が行った機能によって状態を設定します。 |
|  | comment | コメント | 申請待ち |  |
|  | process\_datatime | 実行時間 |  | 各機能の実際の実行時間 |
|  | status\_count | 状態をカウント | １ | 申請したフローに対して申請者が行った機能をカウント |
|  | delete\_flag | 削除フラグ | ０ | １：中止　２：削除 |

## **ビジネスロジック：フロー状態一覧**

クラス:FlowRequest

アクション:FlowApprovalResult（string flow\_request\_id , string user\_id）

// 入力データを整形し、必要な形式に変換

        var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”R1”

approve\_type= “申請フロー作成者“

require\_position=”一般“

approve\_id=user\_id

status=申請する・申請待ち

comment=申請待ち

status\_count=１

delete\_flag = 0

        };

戻り値：flowApprovalResultData

flowApprovalResutlDataをflow\_approval\_resultテーブルにデータを挿入します。

DBに接続処理

## **③機能：保存・申請・中止・削除**

### **保存**

クラス:FlowRequest

アクション:FlowRequestSave（）

帳票に関するデータを保存します。※帳票利用設計書を参照。

### **申請**

クラス:FlowRequest

アクション:FlowRequestApply（string flow\_request\_id , string node\_id）

説明： フローを申請しました。flow\_approval\_resultテーブルのstatusを”申請“に更新します。

更新query

UPDATE flow\_approval\_result

SET status= “申請“, comment = ”SKT001…を申請しました。“

WHERE flow\_request\_id = 'SKT001…' and node\_id = ‘R1’

申請後は申請ボタンが「引き下げ」ボタンに変更されます。後は次の承認者を決定ます。④の承認を参照

### **中止**

クラス:FlowRequest

アクション:FlowRequestStop（string flow\_request\_id , string node\_id）

説明：フロー全体を終了にします。flow\_approval\_resultテーブルのdelete\_flagを１に更新します。

更新query

UPDATE flow\_approval\_result

SET delete　= 1, status\_count = 現状のカウント+1, comment = ”SKT001…を中止しました。“

WHERE flow\_request\_id = 'SKT001…' and node\_id = ‘R1’

### **削除**

クラス:FlowRequest

アクション:FlowRequestDelete（string flow\_request\_id , string user\_id）

説明：削除フラグを更新して一覧から隠すします。flow\_approval\_resultテーブルのdelete\_flagを２に更新します。

更新query

UPDATE flow\_approval\_result

SET delete　= ２, status\_count = 現状のカウント+1、, comment = ”SKT001…を削除しました。“

WHERE flow\_request\_id = 'SKT001…' and node\_id = ‘R1’

## **④機能：承認**

if(user\_leve<8){ 飛び先　＝　A1}

else\_if(user\_leve=8){ 飛び先　＝　A2}

else\_if(user\_leve=10){ 飛び先　＝　A3}

else\_if(user\_leve=12){ 飛び先　＝　A4}

else{ 飛び先＝A5}

申請者

if(最終決裁者＝責任者){ 飛び先＝A10}

else{ 飛び先　＝　A5}

else\_if(user\_leve=10){ 飛び先　＝　A3}

else\_if(user\_leve=12){ 飛び先　＝　A4}

else{ 飛び先＝A5}

if(最終決裁者＝副担当役員)

{ 飛び先＝A6}

else{ 飛び先　＝　A10}

else{ 飛び先　＝　A5}

else\_if(user\_leve=10){ 飛び先　＝　A3}

else\_if(user\_leve=12){ 飛び先　＝　A4}

else{ 飛び先＝A5}

if(最終決裁者＝責任者){ 飛び先＝A10}

else{ 飛び先　＝　A9}

else\_if(user\_leve=10){ 飛び先　＝　A3}

else\_if(user\_leve=12){ 飛び先　＝　A4}

else{ 飛び先＝A5}

if(最終決裁者＝責任者){ 飛び先＝A10}

else{ 飛び先＝A8}

if(修＝修繕){ if (金額>＝50000001){ 飛び先　＝　A11} else{飛び先　＝　A12}}

else{ if (金額>=20000001){ 飛び先　＝　A11} else{飛び先　＝　A12}}

## **ビジネスロジック：承認者決定処理**

関連クラスとアクション：

クラス：FlowApprove

アクション：NextApproverDecision（string flow\_id, string\_node\_id）

説明：承認者決定処理

参照テーブル： flow\_request\_client, flow\_condition\_setting, flow\_logical, flow\_request

1. flow\_request\_clientの　condition\_setting\_flagを取得します。

取得sql (flow\_id=F1, node\_id = R1)

select frc.condition\_setting\_flag from flow\_request\_client frc left join flow\_request fr on fr.flow\_id = frc.flow\_id , fr.node\_id = frc.node\_id

1. condition\_setting\_flag がtrue (R1ノードに条件があること)　の場合だけflow\_condition\_settingテーブルを参照します。
2. コンピューターのスクリーンショット

   AI によって生成されたコンテンツは間違っている可能性があります。

flow\_condtion\_settingテーブルからnode\_id = R1に関する condition\_setting\_idを選び、その中で重複しない値を取得しています。

取得した値は、condition\_setting\_id（例えば、CS1, CS2など）をリスト変数に保存し、使用する準備を備います。

取得sql

var conditionSettingIds = select distinct condition\_setting\_id from flow\_condition\_setting where flow\_id = F1 and node\_id = R1。

condition\_setting\_idをループしてcondition\_idを取得します。

各condition\_setting\_idに関連するcondition\_idをWhere句を使って取得します。

結果として、CS1に関連するcondition\_id、その後にCS2に関連するcondition\_idを貰います。

// condition\_setting\_idに関連するcondition\_idを取得

foreach (var conditionSettingId in ConditionSettingId){

var conditionIds = select \* from flow\_condition\_setting where condititon\_setting\_id = conditionSettingId }

// 各condition\_id

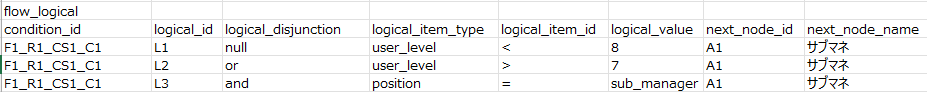
foreach (var conditionId in conditionIds){

1. condition\_idを基にしてflow\_logicalテーブルと紐づけます。詳細の論理和を取得します。

condition\_id（例えば F1\_R1\_CS1\_C1）に基づいてflow\_logicalテーブルから関連するデータを全て取得しています。

var logicalRecords = select \* from flow\_logical where condititon\_id = conditionId }

例：以下のデータを使います。



1. 取得した各レコードに対して、論理演算子や条件式を生成し、最終的な条件式を組み立てます。

生成したい条件式の列　⇒　user\_level < 8 or user\_level > 7 and position = sub\_manager

1. 条件式を構築

string conditionExpression = string.Empty;

bool firstCondition = true;

logicalRecordsをループして各レコードに基づいて条件式を構築

FlowLogicalDataクラスはflow\_logicalテーブルの１行を表します。

logical\_item\_typeとlogical\_valueに基づいて条件を生成

foreach (var record in logicalRecords)

{

GenerateUserLevelCondition と GeneratePositionCondition メソッドは、それぞれ user\_level と position に関する具体的な条件式を生成します。

string currentCondition = string.Empty;

if (record.LogicalItemType == "user\_level")

{

currentCondition =GenerateUserLevelCondition(record);

}

else if (record.LogicalItemType == "position")

{

currentCondition = GeneratePositionCondition(record);

}

logical\_disjunction に基づいて条件を組み合わせ

logical\_disjunction（or や and）の値に応じて、条件式に適切な論理演算子（or や and）を追加しています。

if (!string.IsNullOrEmpty(record.LogicalDisjunction))

{

if (record.LogicalDisjunction == "or")

{

if (!firstCondition)

conditionExpression += " or ";

conditionExpression += currentCondition;

⇒or user\_level > 7

}

else if (record.LogicalDisjunction == "and")

{

if (!firstCondition)

conditionExpression += " and ";

conditionExpression += currentCondition;

⇒and position = sub\_manager

}

}

else

{

currentCondition;

⇒user\_level < 8

}

戻り値：conditionExpression

条件を具体的に生成するメソッド

クラス：GenerateUserLevelCondition（FlowLogical record）

説明：GenerateUserLevelCondition と GeneratePositionCondition メソッドは、それぞれ user\_level と position に関する具体的な条件式を生成します。

GenerateUserLevelCondition(FlowLogical record)

{

string operatorSymbol = record.LogicalItemId; // <, >, = など

string value = record.LogicalValue; // 比較対象の値

return $"user\_level {operatorSymbol} {value}";

結果⇒user\_level < 8

結果⇒user\_level >7

}

GeneratePositionCondition(FlowLogical record)

{

string operatorSymbol = record.LogicalItemId; // = など

string value = record.LogicalValue; // 比較対象の値

return $"position {operatorSymbol} {value}";

結果⇒position = sub\_manager

}

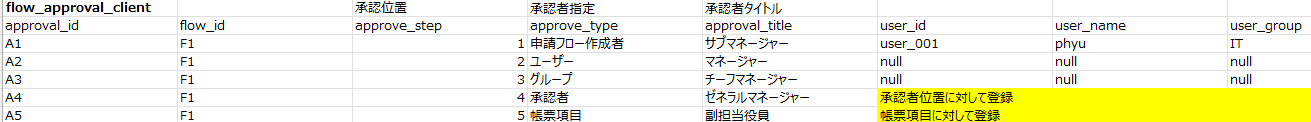
それに従って条件式を組み合わせが完了しましたら条件を基にして承認者を判断します。

userlevel =申請者Userlevel　（ユーザーの設定によって値が変わる）※フロー利用を参照

position = 申請者の役割（ユーザーの設定によって値が変わる）※フロー利用を参照

if(conditionExpression){

※trueの場合はループをbreakしてF1\_R1\_CS1\_C1設定したnext\_node\_id　A1に移動します。}

**ApproveTypeを見て承認依頼とメール通知**

A1の情報をflow\_approval\_clientテーブルからapprove\_typeはユーザーの場合は設定したユーザー情報を取得します。

取得したユーザに承認依頼メールと承認通知を送ります。

approve\_typeはグループの場合は設定したuser\_level\_fromとuser\_level\_toを見ます。そのグループに所属しているユーザー（設定したlevel範囲にいる）全ユーザーに承認依頼メールと承認通知を送ります。

**flow\_approval\_resultテーブルにも登録**します。

var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”A1”

approve\_type= “ユーザー“

require\_position=”sub\_manager“

approve\_id=user\_id

status=承認待ち

comment=flow\_request\_id・・・を承認依頼しました。

status\_count=１

delete\_flag = 0

        };

戻り値：flowApprovalResultData

flowApprovalResutlDataをflow\_approval\_resultテーブルにデータを挿入します。

サブマネージャーが承認をクリックした場合はflow\_approval\_resultのstatusを更新します。

update status = ‘承認‘　, comment = ‘flow\_request\_id・・・を承認しました。’

from flow\_approval\_result where flow\_request = ‘F1’ and node\_id = ‘A1’

承認済の場合は承認ボタンが「引き下げ」ボタンに変更されます。

1. flow\_approval\_clientのA1のcondition\_setting\_flag がfalse の場合そのまま次の承認者に進みます。

Ａ１のapprove\_step　＝１を　＋１にします。apporve\_step２の情報を取得します。

A2情報を貰いました。

1. **ApproveTypeを見て承認依頼とメール通知を参照**
2. **flow\_approval\_resultテーブルにも登録**
3. A2のcondition\_setting\_flag がfalse の場合そのまま次の承認者に進みます。

A2のapprove\_step　＝2を　＋１にします。apporve\_step3の情報を取得します。

A3情報を貰いました。

1. **ApproveTypeを見て承認依頼とメール通知を参照**
2. **flow\_approval\_resultテーブルにも登録**
3. A3のcondition\_setting\_flag がfalse の場合そのまま次の承認者に進みます。

A3のapprove\_step　＝3を　＋１にします。apporve\_step4の情報を取得します。

A4情報を貰いました。

1. **ApproveTypeを見て承認依頼とメール通知を参照**
2. **flow\_approval\_resultテーブルにも登録**
3. A4のcondition\_setting\_flag がtrue の場合はNextApproverDecision関数を呼び出します。**（NextApproverDesicionを参照）**
4. A4 の最終決裁者　は責任者の場合は　A10に移動します。ではない場合はA5に移動します。

残ったノードも上記の処理の通り行います。

flow\_approval\_client テーブルから flow\_id ごとに最大の approve\_step を取得できます。

SELECT flow\_id, MAX(approve\_step) AS max\_approve\_step

FROM flow\_approval\_client

GROUP BY flow\_id;

最大apporve\_stepと現在のapprove\_step同じの場合はフローを終了します。

flow\_approval\_resultテーブルに登録します。

## **機能：引き上げ**

if(user\_leve<8){ 飛び先　＝　A1}

else\_if(user\_leve=8){ 飛び先　＝　A2}

else\_if(user\_leve=10){ 飛び先　＝　A3}

else\_if(user\_leve=12){ 飛び先　＝　A4}

else{ 飛び先＝A5}

理事

副理事

執行役員

ゼネ

チーフ

マネ

サブマネ

申請者

R1のユーザーレベルが＜８の場合は引き上げノードは？

A1が承認、通知とメールも送る。

A2,A3,A４が引き上げ承認できた。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝８の場合は引き上げノードは？

A2が承認、通知とメールも送る。

A3,A４が引き上げ承認できた。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝10の場合は引き上げノードは？

A3が承認、通知とメールも送る。

A４が引き上げ承認できた。。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝12の場合は引き上げノードは？

A4が承認、通知とメールも送る。引き上げ承認はユーザー設定を基にして判断。ユーザー設定は責任者権限なら責任者権限以上役員は引き上げ承認は行えません。

飛び先A5の場合は引き上げノードは？

そのノードの上位ノードに引き上げできるかはユーザー設定を見る。

R1のユーザーレベルが＜８の場合は引き下げノードは？

A1が承認前だったらR1前のアクションした方に戻る。前の方がやったアクションを取り消す。もう一度承認できる状態に戻る。

R1のユーザーレベルが＝８の場合は引き下げノードは？

A1が承認前だったらR1前のアクションした方に戻る。前の方がやったアクションを取り消す。もう一度承認できる状態に戻る。

R1のユーザーレベルが＝10の場合は引き下げノードは？

A1が承認前だったらR1前のアクションした方に戻る。前の方がやったアクションを取り消す。もう一度承認できる状態に戻る。

R1のユーザーレベルが＝12の場合は引き下げノードは？

A1が承認前だったらR1前のアクションした方に戻る。前の方がやったアクションを取り消す。もう一度承認できる状態に戻る。

飛び先A5の場合は引き下げノードは？

A1が承認前だったらR1前のアクションした方に戻る。前の方がやったアクションを取り消す。もう一度承認できる状態に戻る。

## **ビジネスロジック：引き上げ承認**

クラス:FlowApprove

アクション:ApproveIncrease（string flow\_request\_id, string current\_approve\_node ）

説明：フローには定義されているが申請においてその時点でアクション権限を持たない上位ノード。

　　この条件に合致するUserが行う「承認」を指す。

　　　　尚。引き上げを行ったUserとの間に存在するフロー(ノード及び条件)に沿ったアクション権限を保持する予定だったノードは「引き上げ承認」のステータスを保持する。

フローを利用する側の flow\_approval\_client の approve\_increase が可能で、かつ authorization\_increase が責任者権限である場合に、引き上げ承認を実行します。

**５の引き上げ承認図を参照**

「引き上げ承認」の対象は、申請者のユーザーレベルが8未満の場合、飛び先はA1（サブマネージャー）となります。責任者権限はゼネラルマネージャーまでに設定されています。ゼネラルマネージャーのノードまで引き上げ承認が設定されます。

まず、ゼネラルマネージャーの approve\_step を判断するために、次のSQLクエリで取得します。

取得sql

int approveIncrease =SELECT approve\_step

FROM flow\_approval\_client

WHERE approve\_title = 'ゼネラルマネージャー'

次に、引き上げ承認を行うノードリストを取得し、リスト変数 authorizationIncrease に格納します。

取得sql

var authorizationIncrease = new List<flow\_approval\_client>();

authorizationIncrease = select \* from flow\_approval\_client where approval\_id <> current\_approve\_node and approve\_step >= @ approveIncrease　and flow\_id = @flow\_id

authorizationIncrease に格納されたリストをループ処理し、以下のように処理します。

approve\_type が「ユーザー」の場合、設定したユーザー情報を取得し、該当ユーザーの画面に引き上げ承認ボタンを表示します。

approve\_type が「グループ」の場合、設定された user\_level\_from と user\_level\_to を基に、その範囲に所属するユーザー全員の画面に引き上げ承認ボタンを表示します。

また、**flow\_approval\_resultテーブルにも登録**します。

var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”A2”

approve\_type= “ユーザー“

require\_position=”マネ“

approve\_id=user\_id

status=引き上げ承認

comment=引き上げ承認が設定されました。

status\_count=１

delete\_flag = 0

        };

この処理は、チーフマネージャーやゼネラルマネージャーのためにも同様に行います。

メモです。

R1のユーザーレベルが＜８の場合は引き上げノードは？

A1が承認、通知とメールも送る。

A2,A3,A４が引き上げ承認できた。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝８の場合は引き上げノードは？

A2が承認、通知とメールも送る。

A3,A４が引き上げ承認できた。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝10の場合は引き上げノードは？

A3が承認、通知とメールも送る。

A４が引き上げ承認できた。。画面に引き上げボタンが表示され。通知とかメールを送らない。

R1のユーザーレベルが＝12の場合は引き上げノードは？

A4が承認、通知とメールも送る。

引き上げ承認はユーザー設定を基にして判断。

デフォルトは責任者権限までです。責任者権限以上役員は引き上げ承認は行いません。

飛び先A5の場合は引き上げノードは？

ユーザーがどのレベルまでの役員を設定したか判断。その設定した役員まで引き上げ承認が行います。

## **機能：引き下げ承認**

クラス：FlowApprove

アクション：ApproveDecrease(string flow\_request\_id, string current\_node\_id)

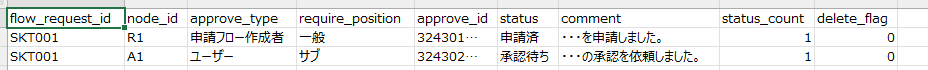
説明：フローに沿ってアクション権限を保持し「承認」や「差戻」のアクションを行ったUserがそのアクションを取り消すこと。尚。引き下げの対象となるのは「承認」及び「差戻」この2つのアクションのみであり「引き上げ」は含まれない為、取り消せない。

## **ビジネスロジック：引き下げ機能**

**５の引き上げ承認図を参照**

「引き上げ承認」の対象は、申請者のユーザーレベルが8未満の場合、飛び先はA1（サブマネージャー）となります。

まずはflow\_approval\_resultテーブルを参照します。

****

・A1のstatusを判断sql

select status from flow\_approval\_result where flow\_request\_id = @flow\_request\_id and node\_id = @current\_node\_id

・A1のstatusが承認待ち（まだ未承認）だったらR1前のアクションした方に戻ります。承認済の場合は取り消せない。

・R1のユーザー画面に最初状態「承認」できる状態になります。

・また、**flow\_approval\_resultテーブルにも登録**します。

var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”R1”

approve\_type= “ユーザー“

require\_position=”一般“

approve\_id=user\_id

status=引き下げ承認

comment=引き下げ承認が設定されました。

status\_count=現状のカウント+1

delete\_flag = 0

        };

**・**引き続きの流れは承認機能の通りです。　**④機能承認を参照**

## **機能：差し戻し**

クラス：FlowApprove

アクション：RevertApprove（string flow\_id, string current\_node\_id）

説明：フローに沿ってアクション権限を保持しているUserが「差戻」として特定のノードを事前若しくは当時に指定してアクション権限を遷移させる。(上位ノード可)このことを表す。

## **ビジネスロジック：差し戻し**

・まずはflow\_approval\_client テーブルを参照します。current\_node\_idの差戻の対象者(承認位置)を判断します。承認位置を基にして承認者情報を取得します。

select \* from flow\_approval\_client where (select send\_back\_step,send\_back\_noti\_flag,send\_back\_noti\_type from flow\_approval\_client where flow\_id = @flow\_id and node\_id = @current\_node\_id)

・差戻の対象者画面に最初状態「承認」できる状態になります。

・また、**flow\_approval\_resultテーブルにも登録**します。

var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”対象ノード”

approve\_type= “対象ユーザー“

require\_position=”一般“

approve\_id=user\_id

status=差戻

comment=差戻が設定されました。

status\_count=現状のカウント+1

delete\_flag = 0

        };

・引き続きの流れは承認機能の通りです。　**④機能承認を参照**

## **機能：否認**

クラス：FlowApprove

アクション：denyApprove(string flow\_request\_id, string current\_node\_id)

説明：フローに沿ってアクション権限を保持しているUserが「否認」を行い、稟議を終了させることを示す。

## **ビジネスロジック：否認**

flow\_approval\_resultテーブルを更新します。

var flowApprovalResultData= new FlowApprovalRequestData

        {

flow\_request\_id= flow\_request\_id

node\_id=”対象ノード”

approve\_type= “対象ユーザー“

require\_position=”一般“

approve\_id=user\_id

status=否認

comment=。。。を否認しました。

status\_count=現状のカウント+1

delete\_flag = 2

        };

## **機能：一括承認**

クラス：FlowApprove

アクション：ApproveBulkRequest(list<string>approveBulkRequet)

説明：同時で複数を帳票を承認します。

## **ビジネスロジック：一括承認**

approveBulkrequestを最後までループします。各ループの中に承認処理を行います。

機能:承認を参照

## **機能：フロー参照（進捗）**

クラス：FlowApprove

アクション：CheckProgress(string flow\_request\_id)

説明： フローの進捗状況や承認者の承認日時などを確認できます。

# **ビジネスロジック：フロー参照（進捗）**

対象のflow\_request\_idの情報をflow\_approval\_resultから取得して画面に表示します。

select \* from flow\_approval\_result where flow\_request\_id = @flow\_request\_id