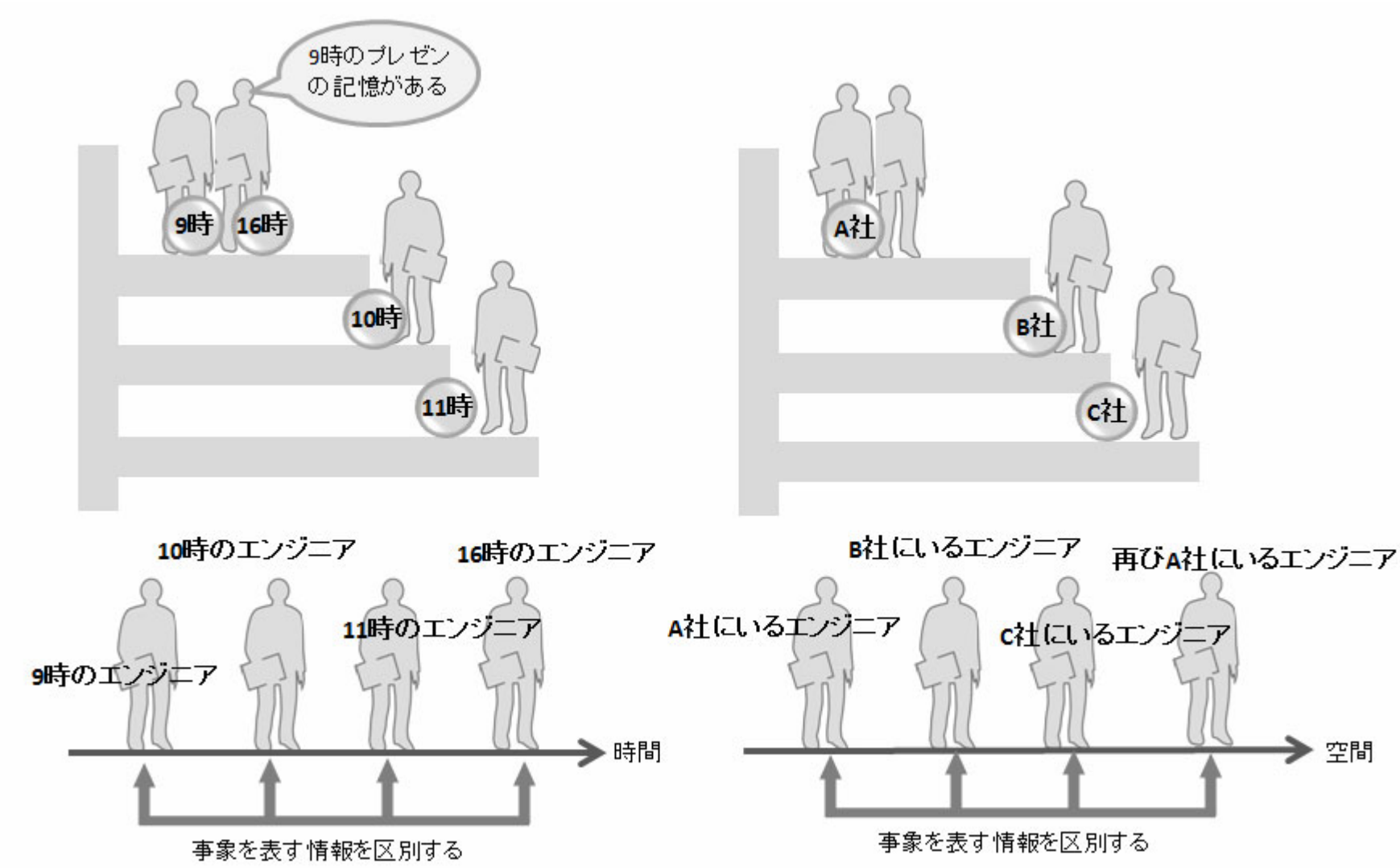
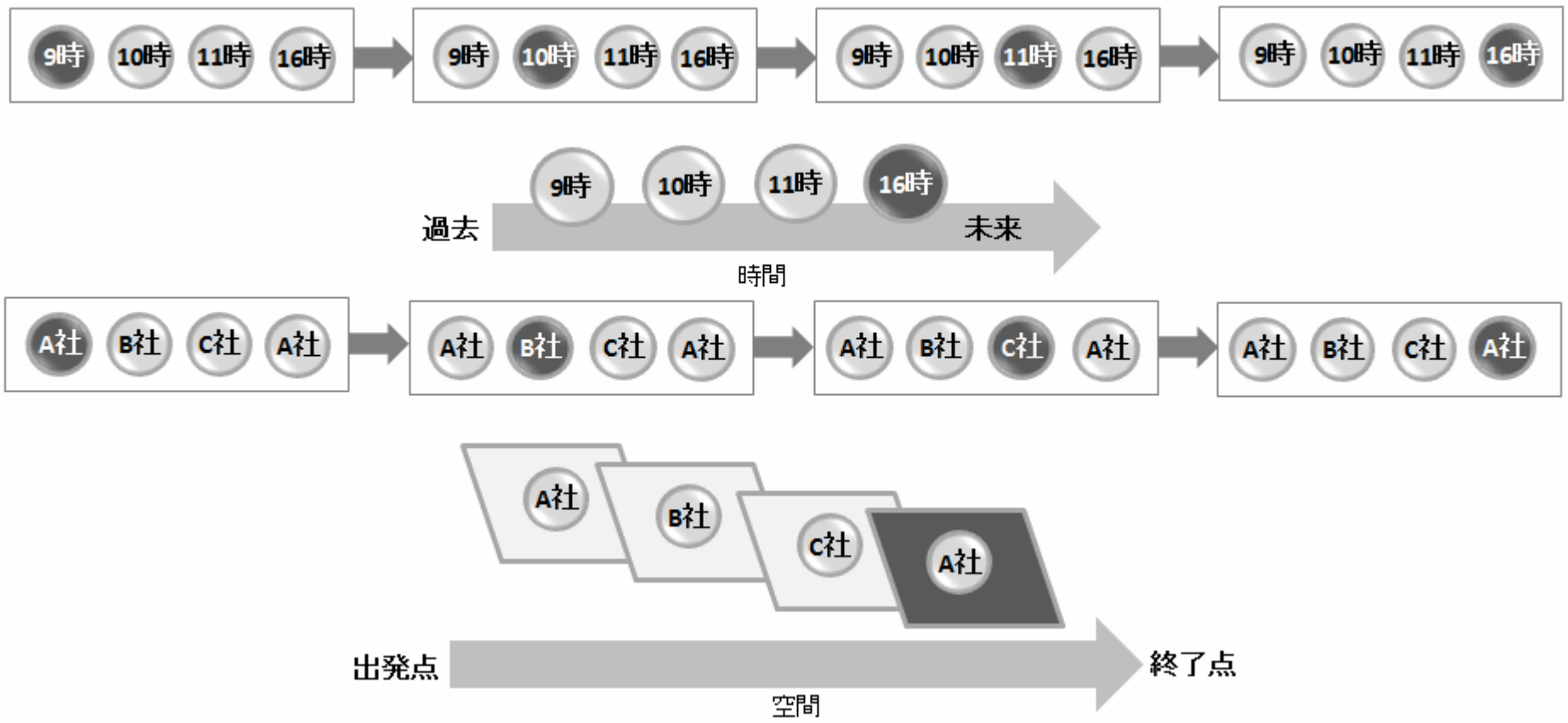
第１章の図表

●図 1-1 我々は，重なり合わない現象の間に境界線を引き，現象を表す情報を区別している

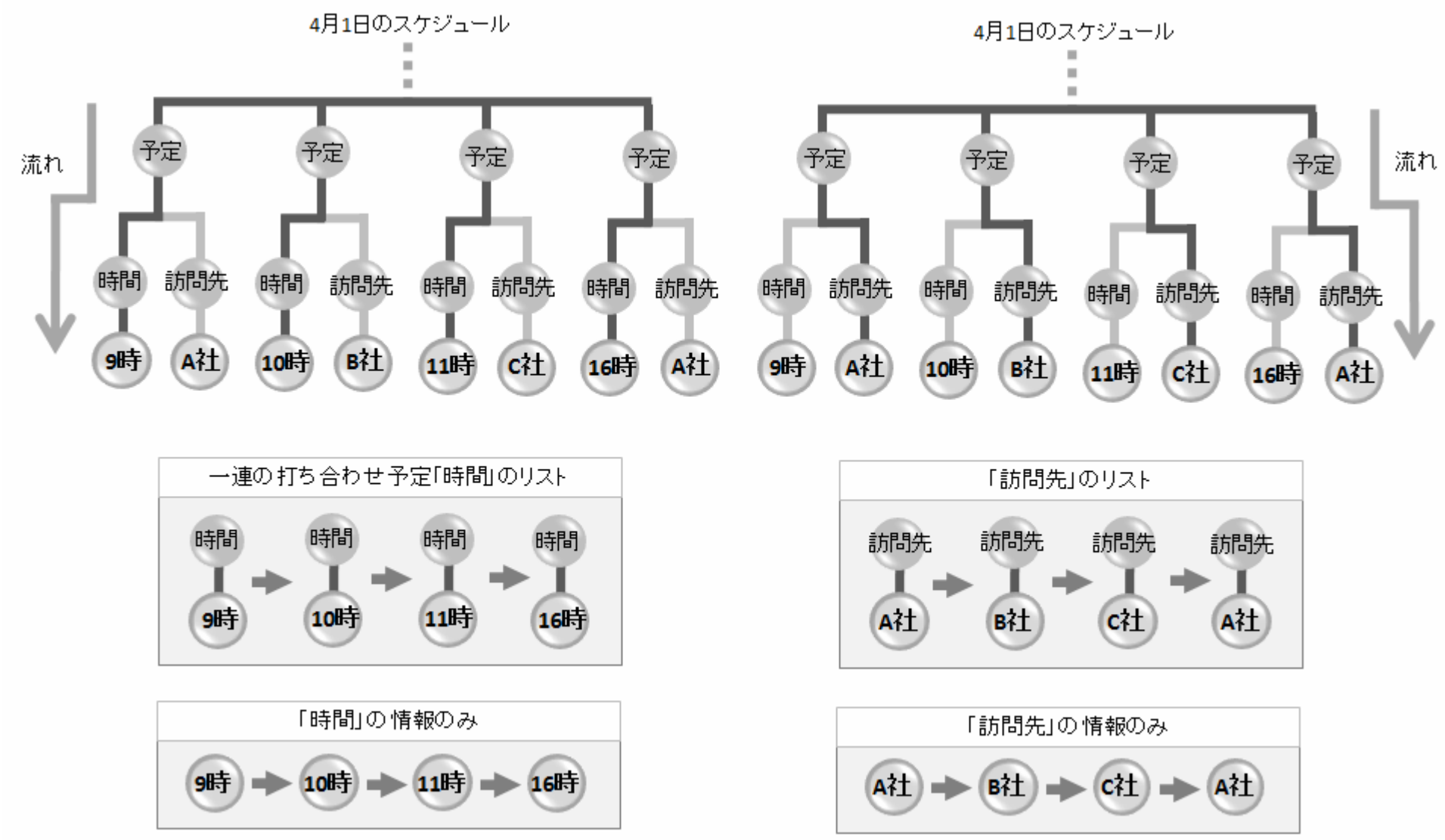


●図 1-2 ある IT エンジニアの訪問時間と訪問先の情報。時間や空間の流れはなく静的である。

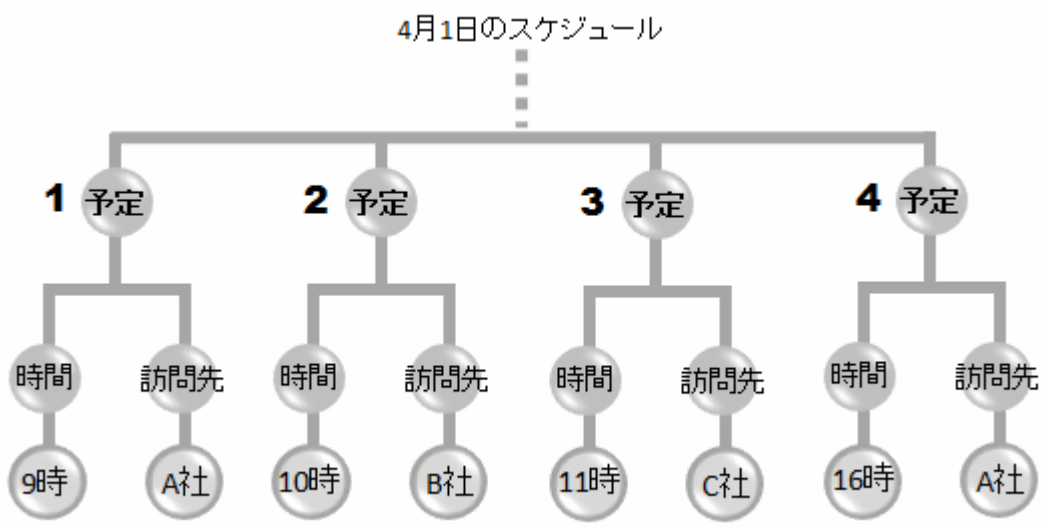
●図 1-3 静的な情報も，観る者がいると，流れができる。



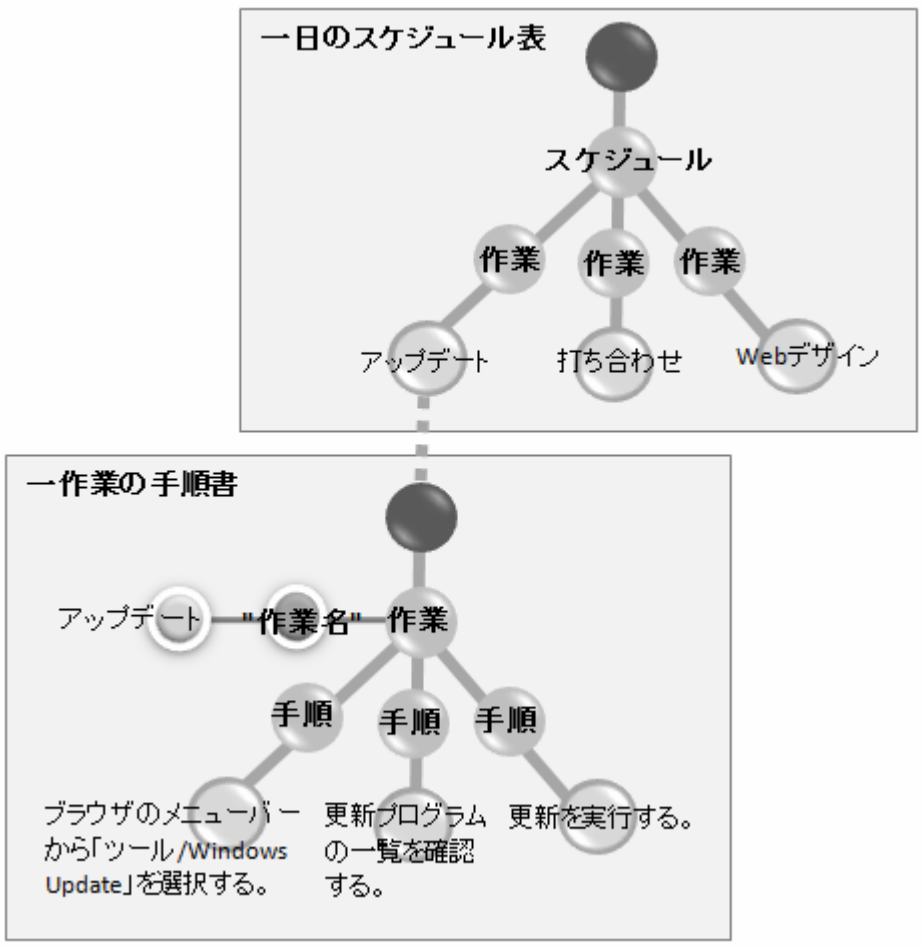
●図 1-4 「流れ」のある情報の部分にも「流れ」がある



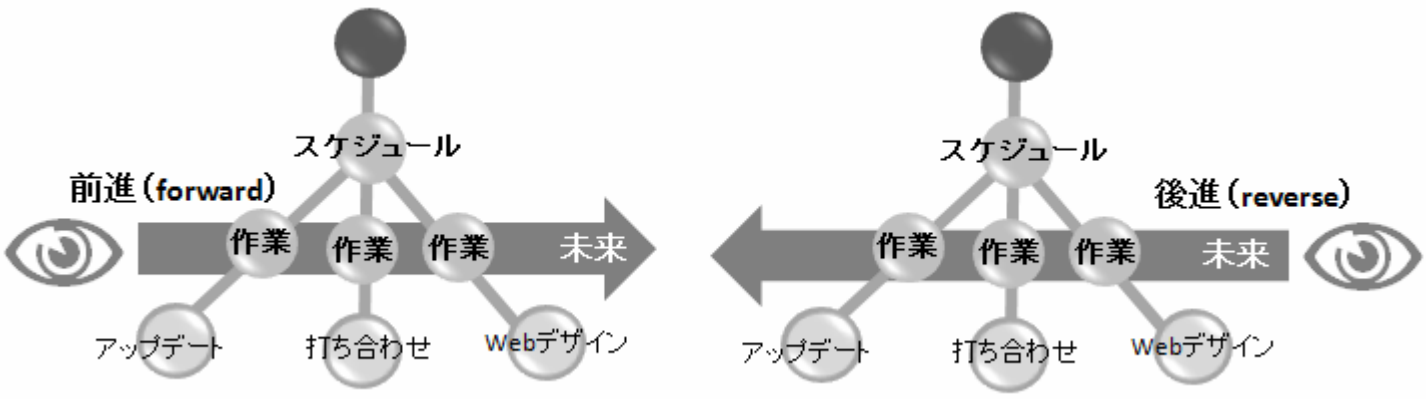
●図 1-5 流れには，方向と順序を持つシーケンスがある



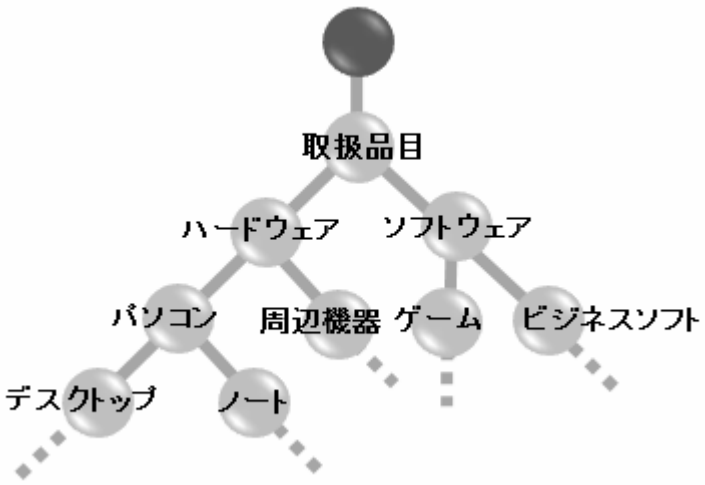
●図 1-6　スケジュール表と，操作手順を表した図の一例



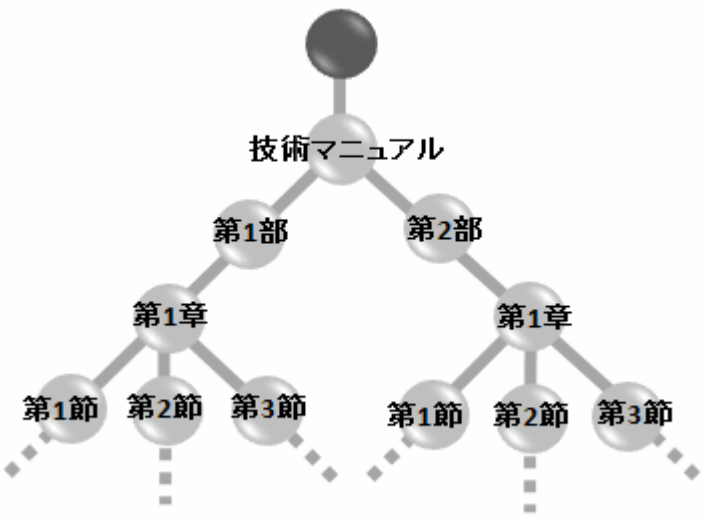
●図 1-7　視点によって，流れの方向は変わる。



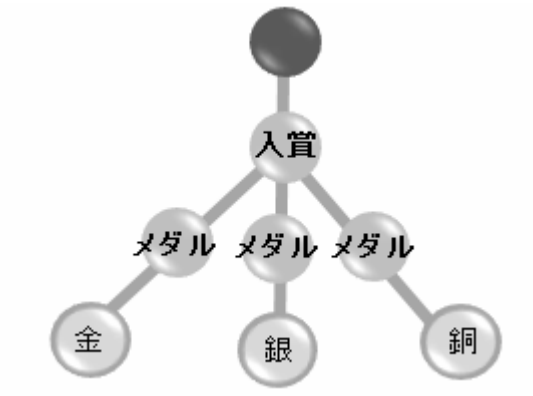
●図 1-8　絞り込み検索には，「外から内」への流れがある。



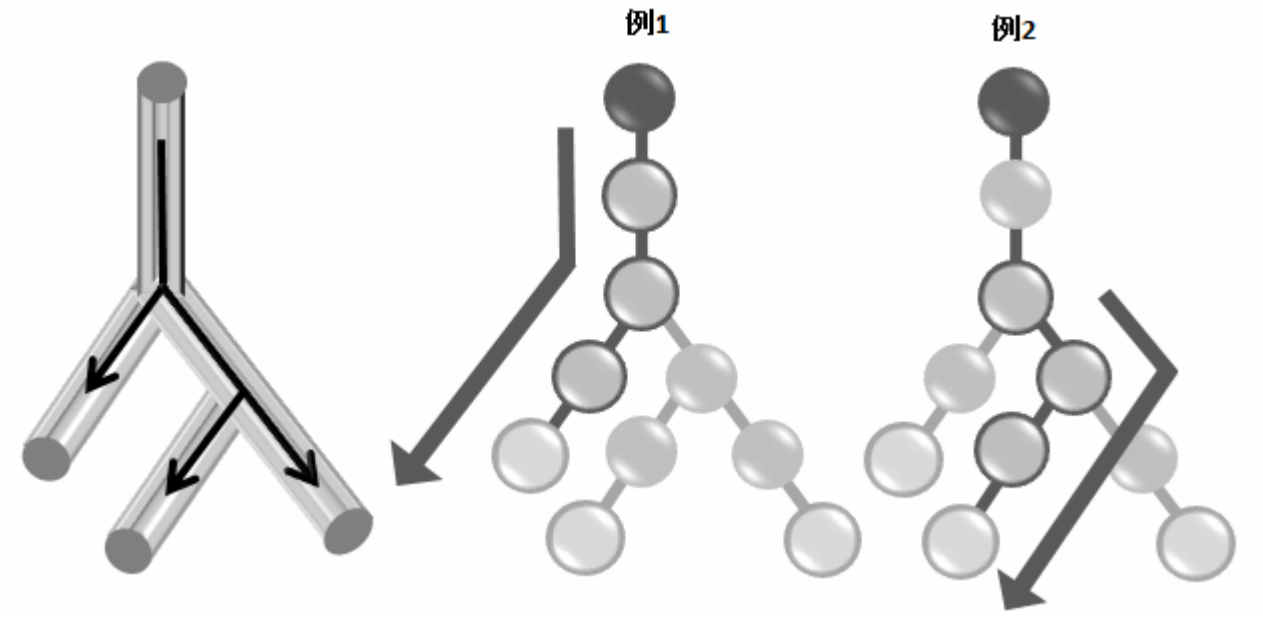
●図 1-9　マニュアルには，「外から内」「前から後」といった 2 種類のプライオリティの流れがある



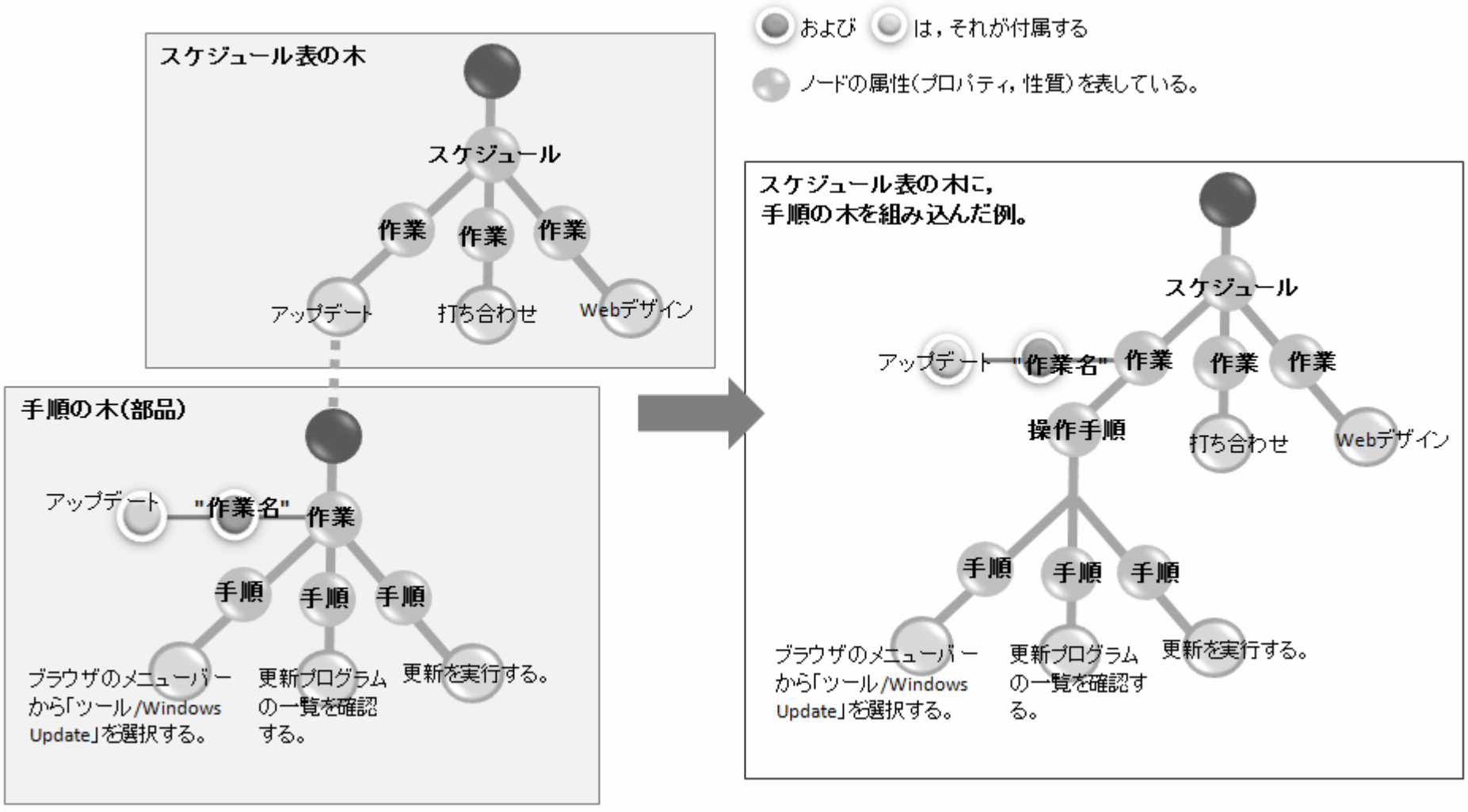
●図 1-10　社会的な「高→低」へのプライオリティによる流れがある。



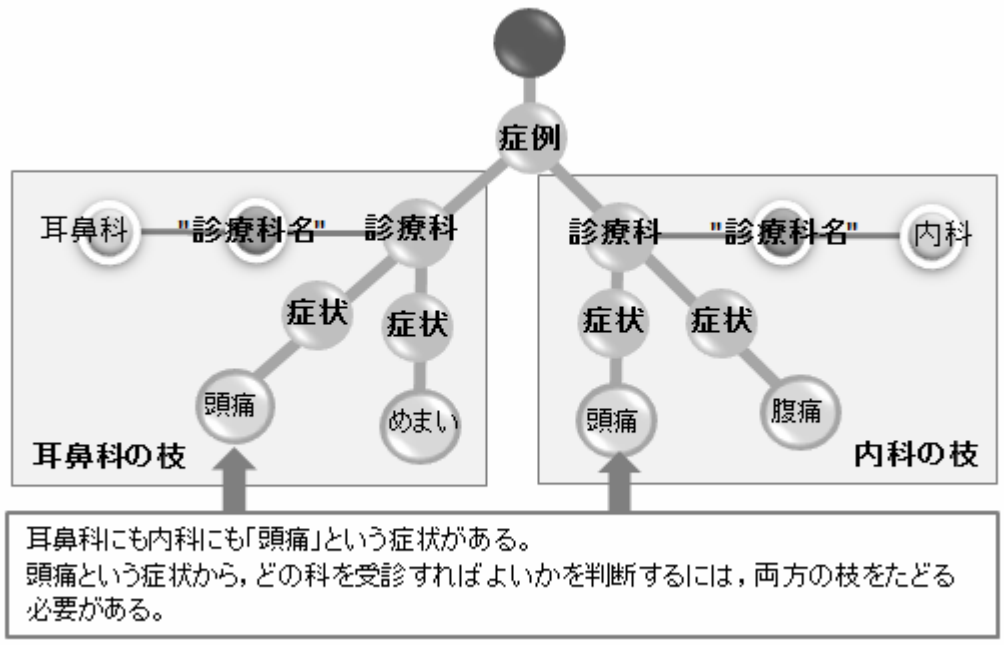
●図 1-11　木構造では流れを表現することができる。



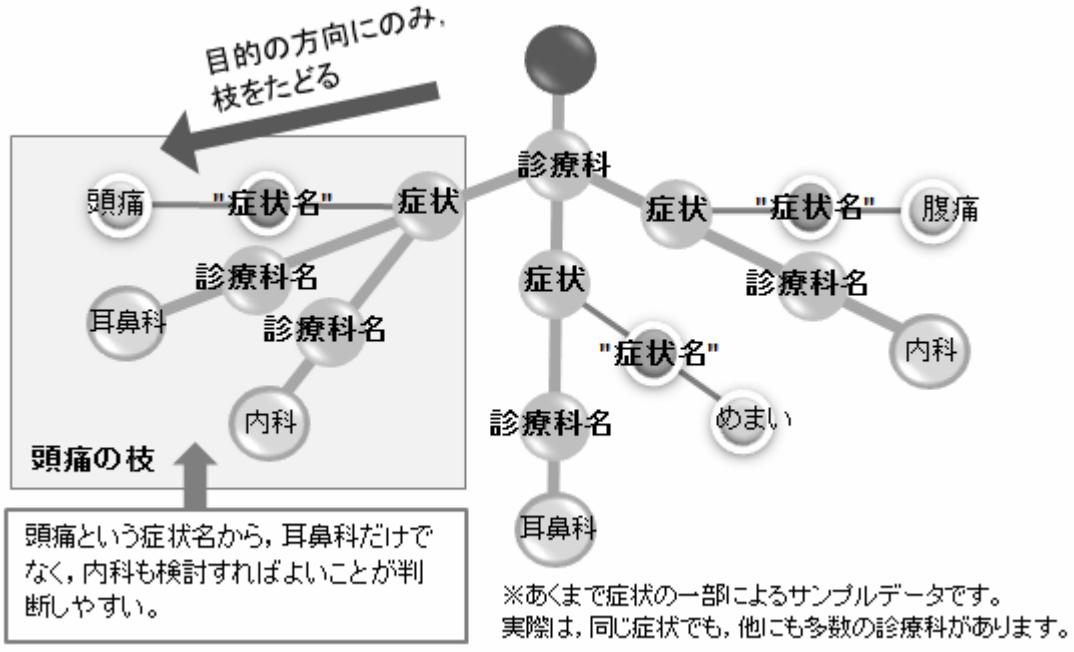
●図 1-12　●図 1-6 のスケジュール表と，操作手順は別の流れを持つが，組み込むこともできる。



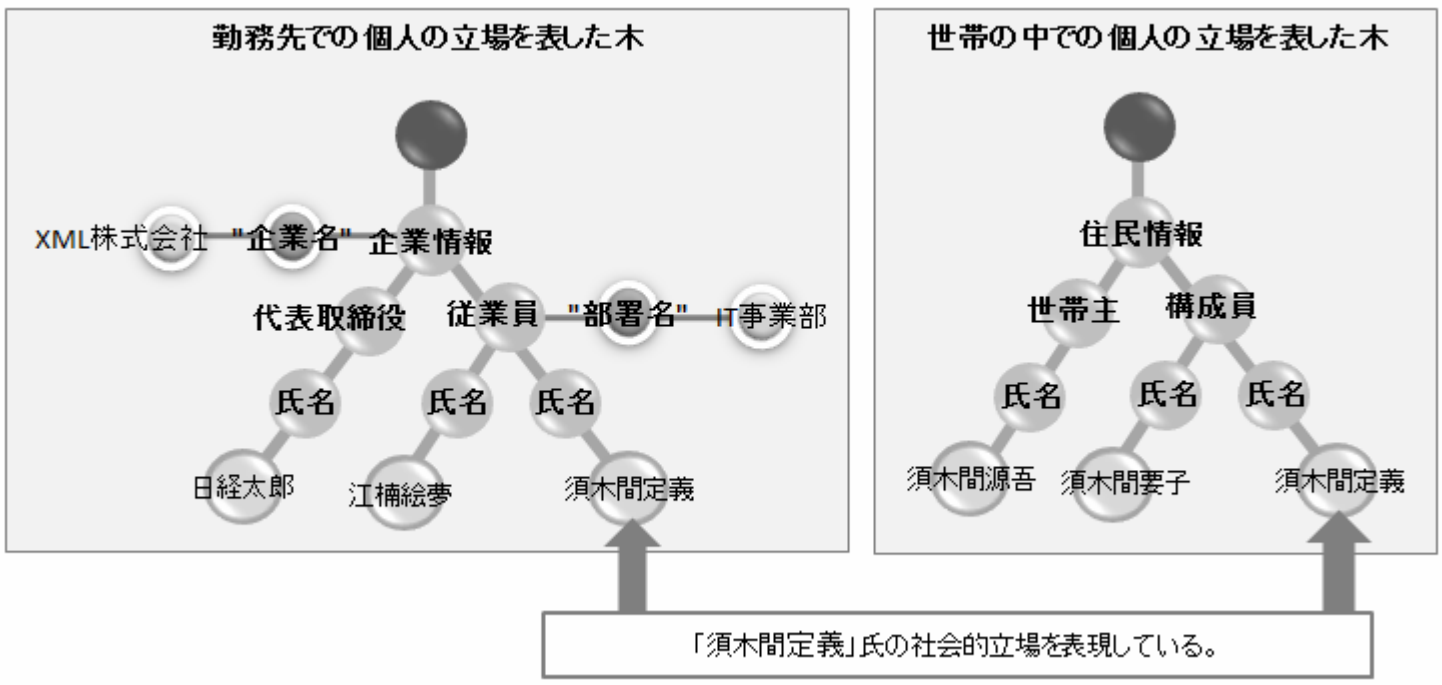
●図 1-13　診療科別分類で，症状を表現した木の例。



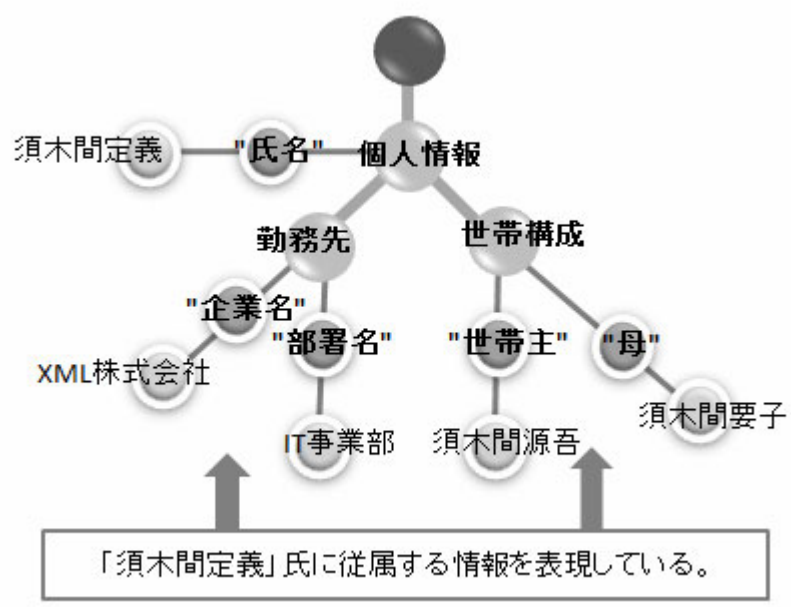
●図 1-14　症状別分類で，診療科名を表現した木の例。



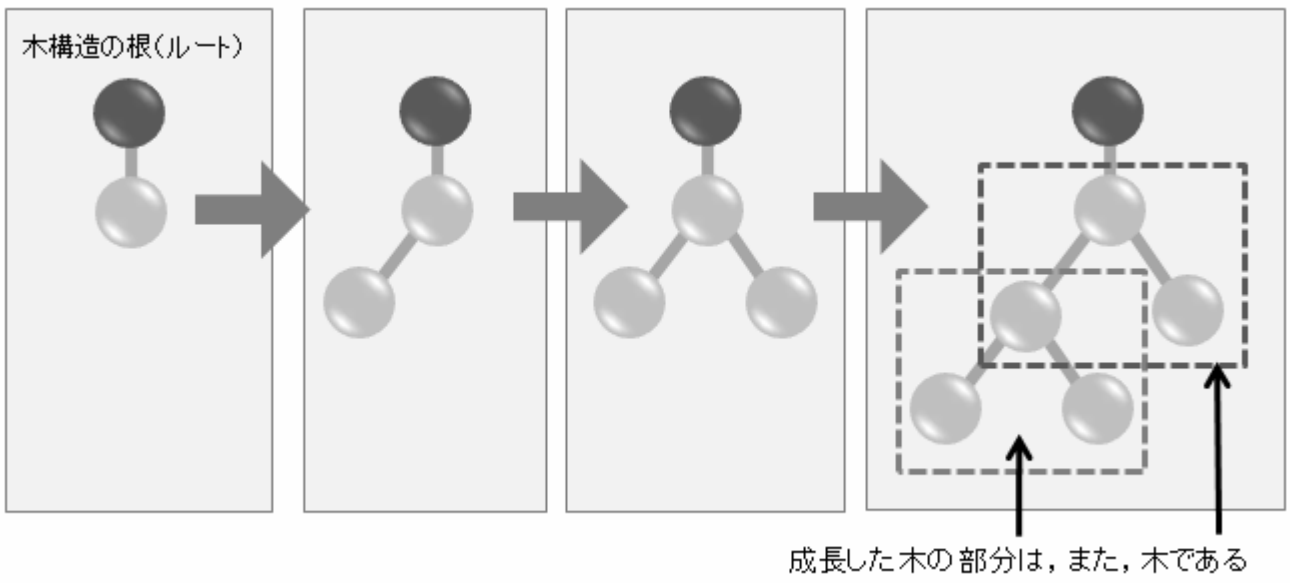
●図 1-15 社会的役割のプライオリティが個人より高い場合の，2 本の木。



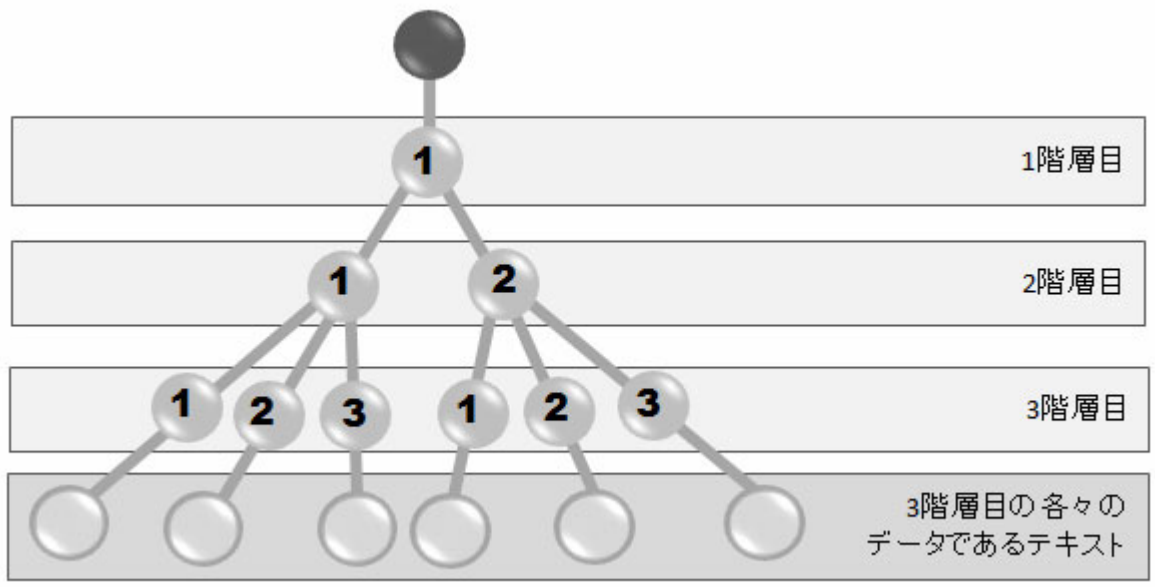
●図 1-16 個人のプライオリティが帰属先より高い場合の木。



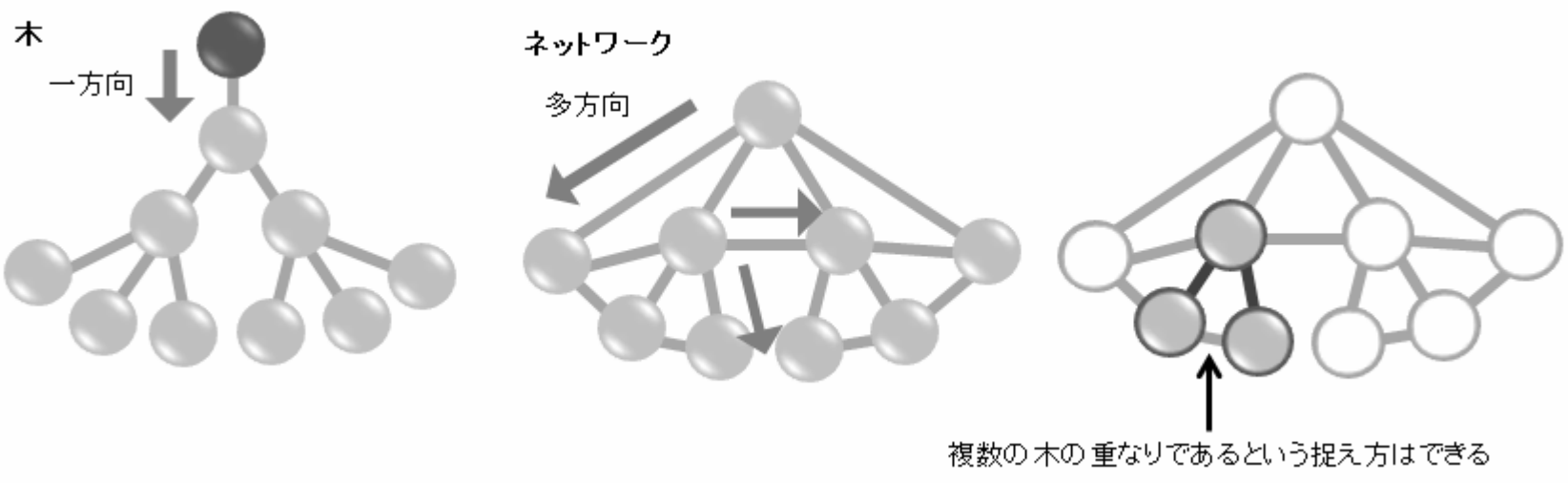
●図 1-17 根（ルート）から生えたノードから木は成長していく



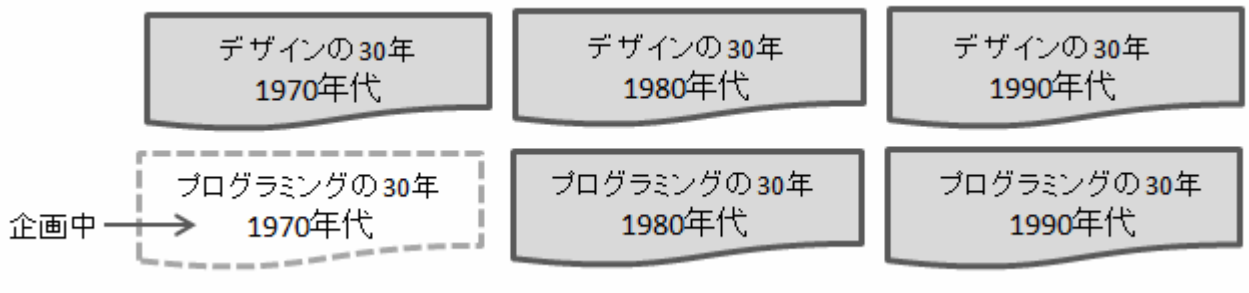
●図 1-18　木では，どのノードも順序指定により一意なキーとして扱える



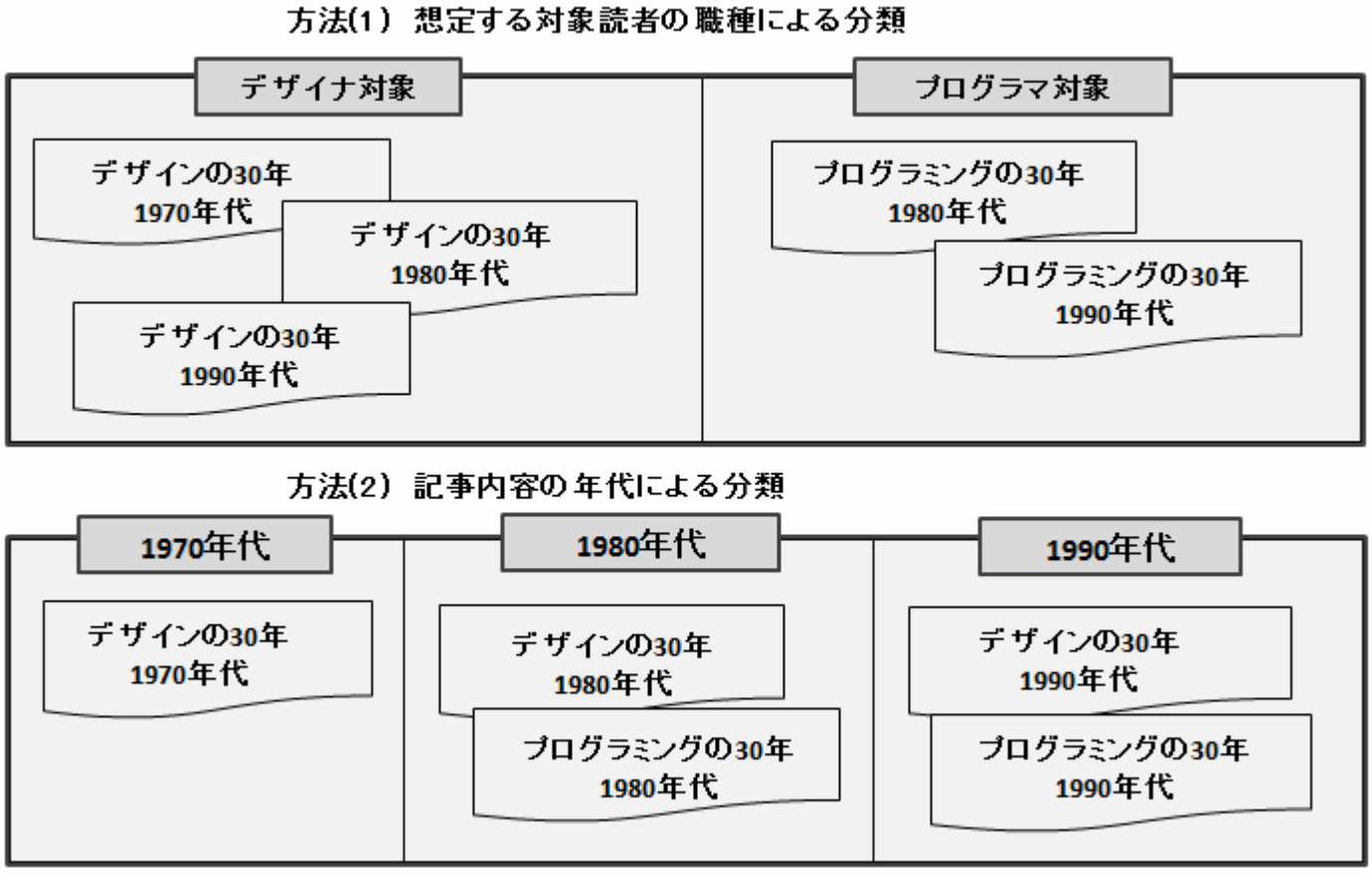
●図 1-19　木とネットワークの違いの一例。木のシーケンスは一方向である



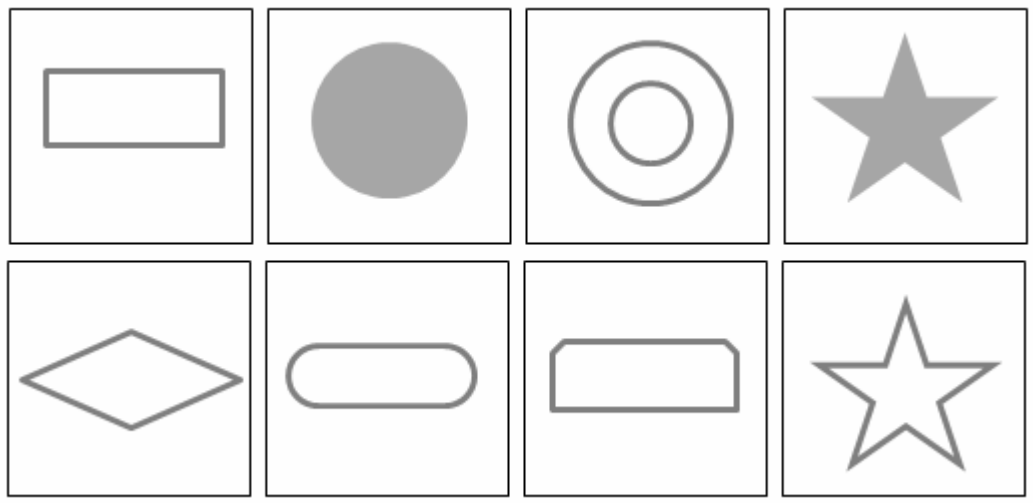
●図 1-20　5 本のオンライン記事



●図 1-21　●図 1-20 の 5 本の記事を分類する基本的な 2 つの方法



●図 1-22　対象の持つ「意味」が異なれば，ヒトによって木構造の判断は異なる



●図 1-23　●図 1-3 の「職種による分類」を表形式で表現した例

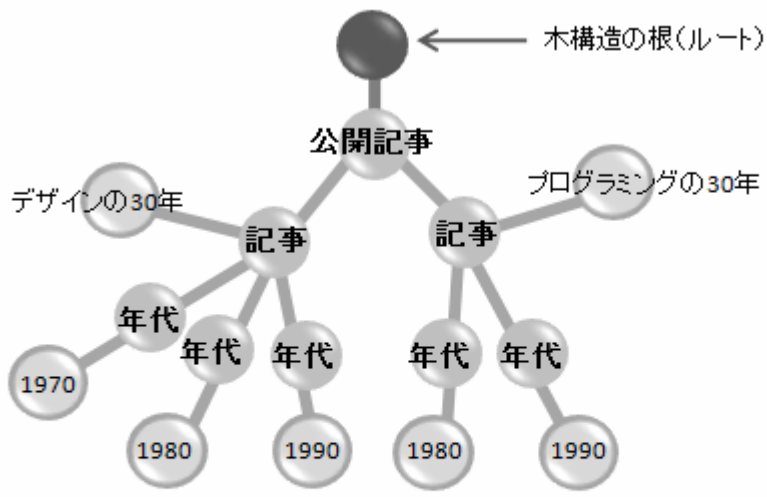
公開記事データベース

ページID	記事	年代
D1970	デザインの30年	1970
D1980	デザインの30年	1980
D1990	デザインの30年	1990
P1980	プログラミングの30年	1980
P1990	プログラミングの30年	1990

デザイナ対象のオンライン記事

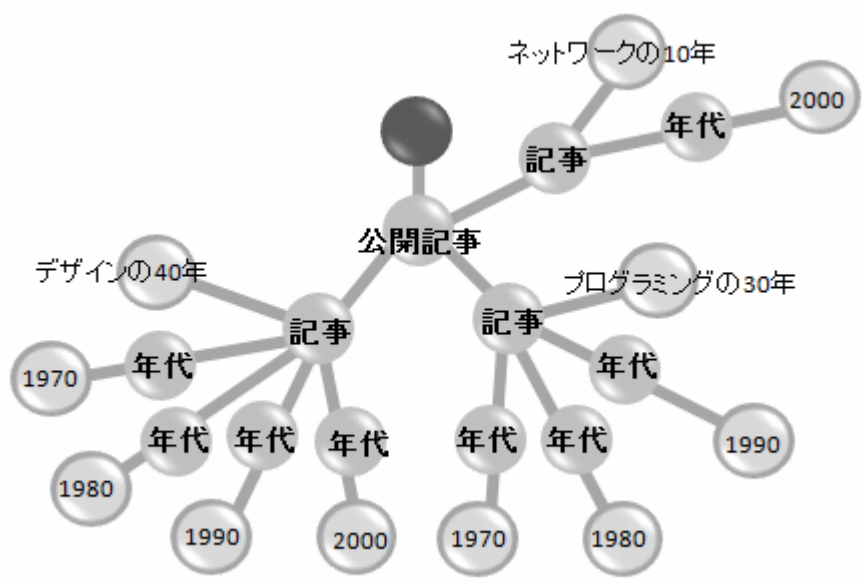
プログラマ対象のオンライン記事

●図 1-24　●図 1-23 の「職種による分類」を木構造で表現した例



※表1-1を忠実に置き換えるのであれば，各「年代」から「ページID」の枝を伸ばすことは可能であるが，木構造では，各「年代」の記事自体が一意であるため，省略している。一意である理由は後述している。

●図 1-25　●図 1-24 の木に，枝を接ぎ木してタイトルのデータを変更した例

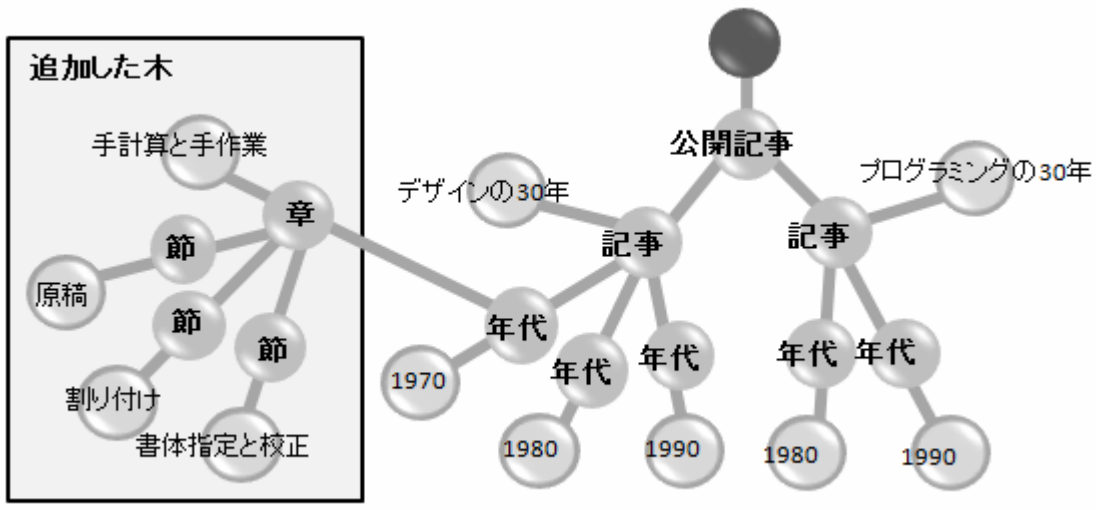


●図 1-26　●図 1-23 にフィールドを追加した例

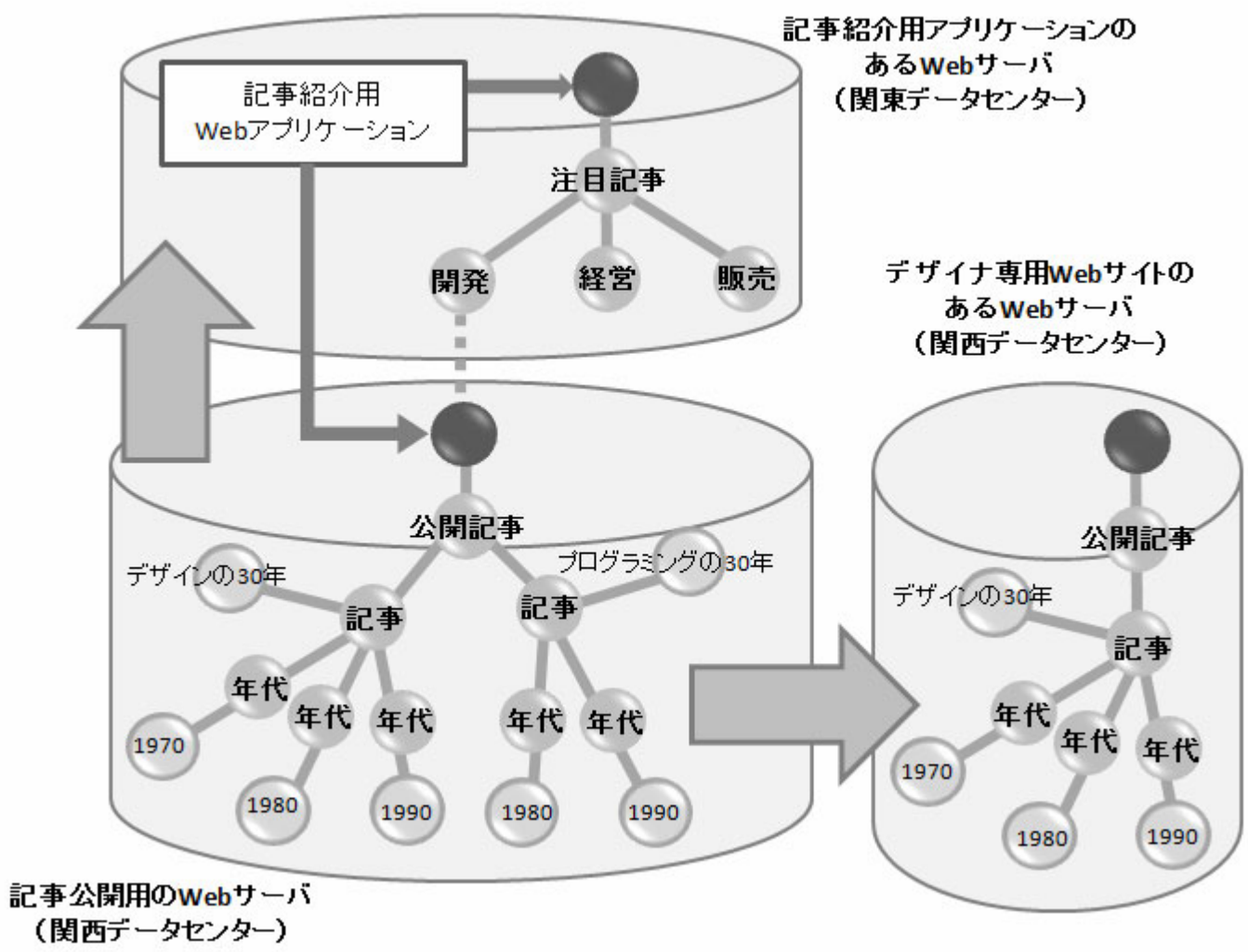
回顧録テーブル

ページID	記事	年代	章	節
D1970P1	デザインの30年	1970	手計算と手作業	原稿
D1970P2	デザインの30年	1970	手計算と手作業	割り付け
D1970P3	デザインの30年	1970	手計算と手作業	書体指定と校正
D1980P1	デザインの30年	1980		
D1990P1	デザインの30年	1990	データがない	
P1980P1	プログラミングの30年	1980		
P1990P1	プログラミングの30年	1990		

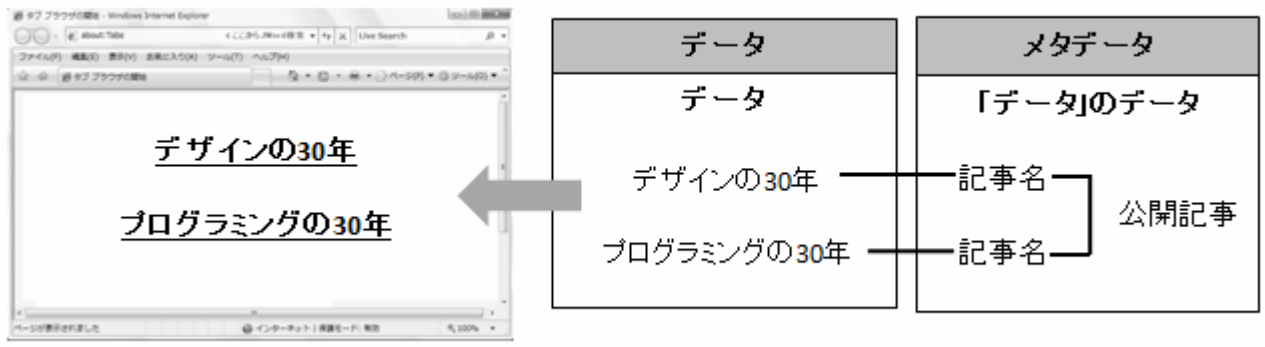
●図 1-27　●図 1-26 を木構造で表現した例



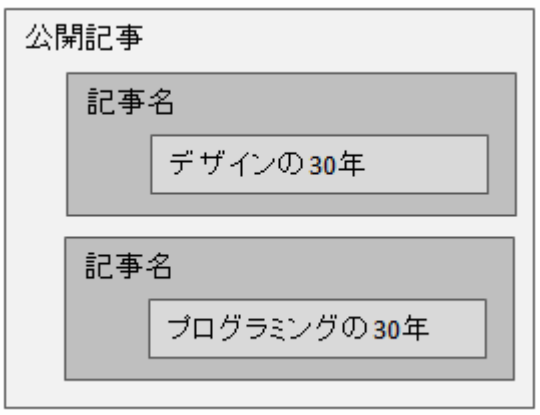
●図 1-28　木は，利用される時間と空間に応じる形で，柔軟に変化する



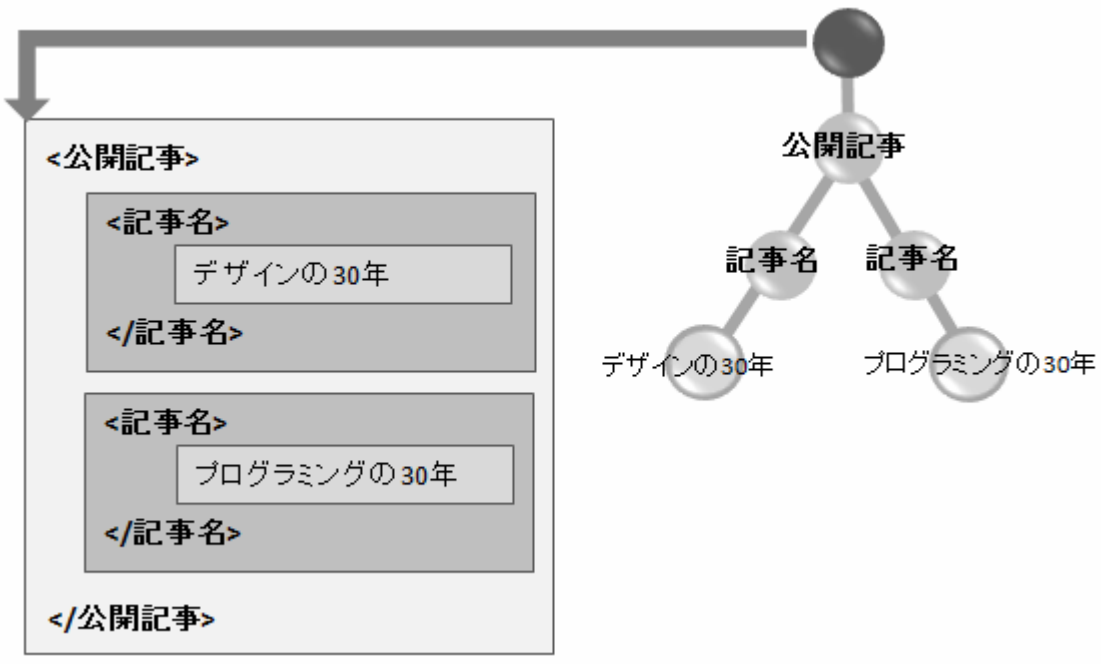
●図 1-29　データとメタデータ。



●図 1-30　●図 1-29 の階層構造は入れ子になっている。

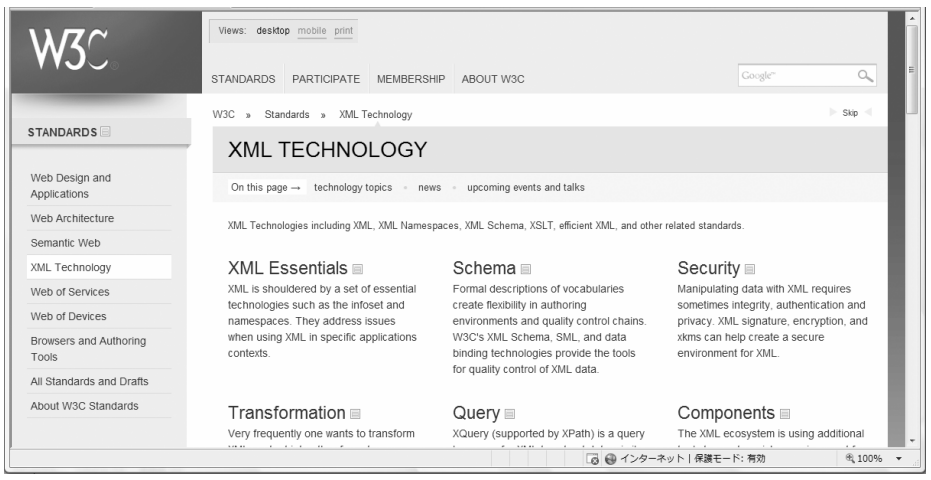


●図 1-31　●図 1-30 のメタデータを<と>で挟んでデータと区別した例と，その木構造。

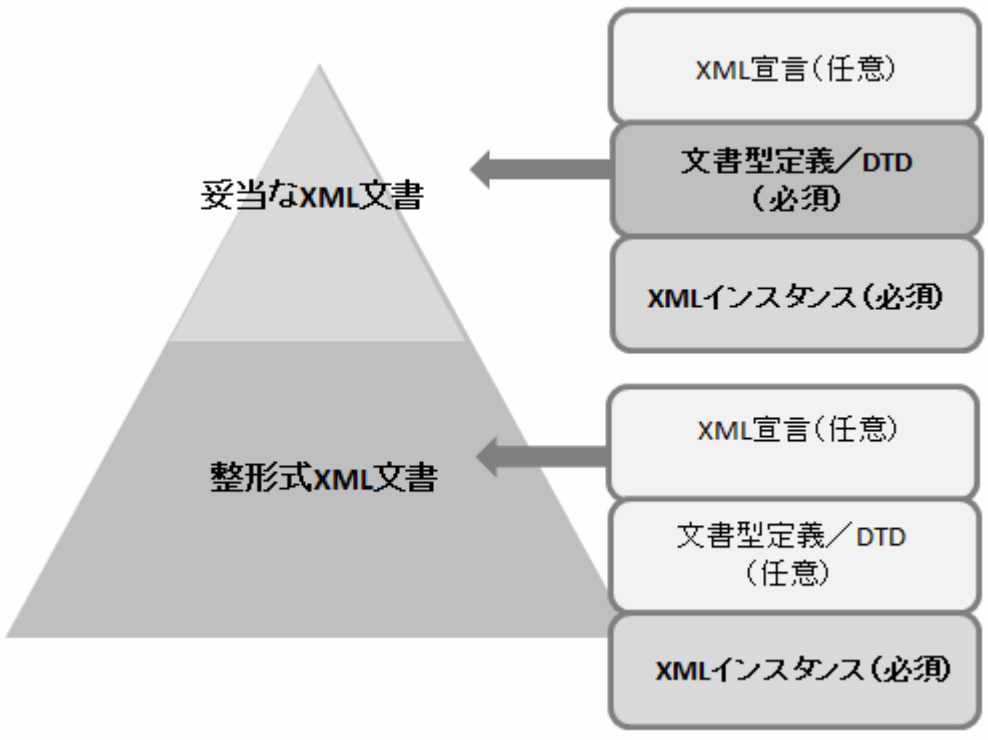


第 2 章の図表

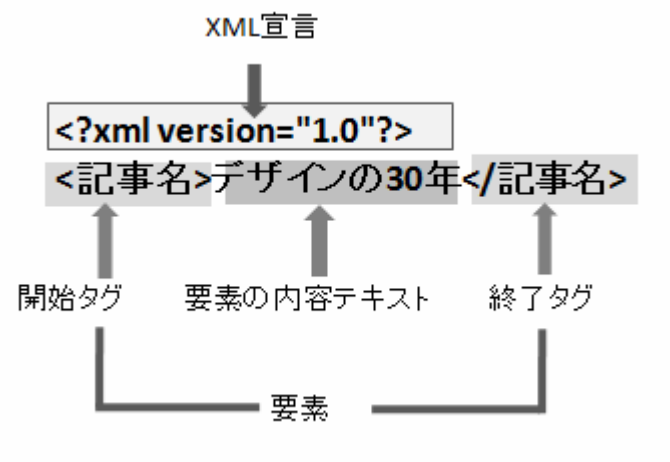
●図 2-1　W3C の XML ページ。仕様書は「XML Specifications」「Translations」コーナから入手できる



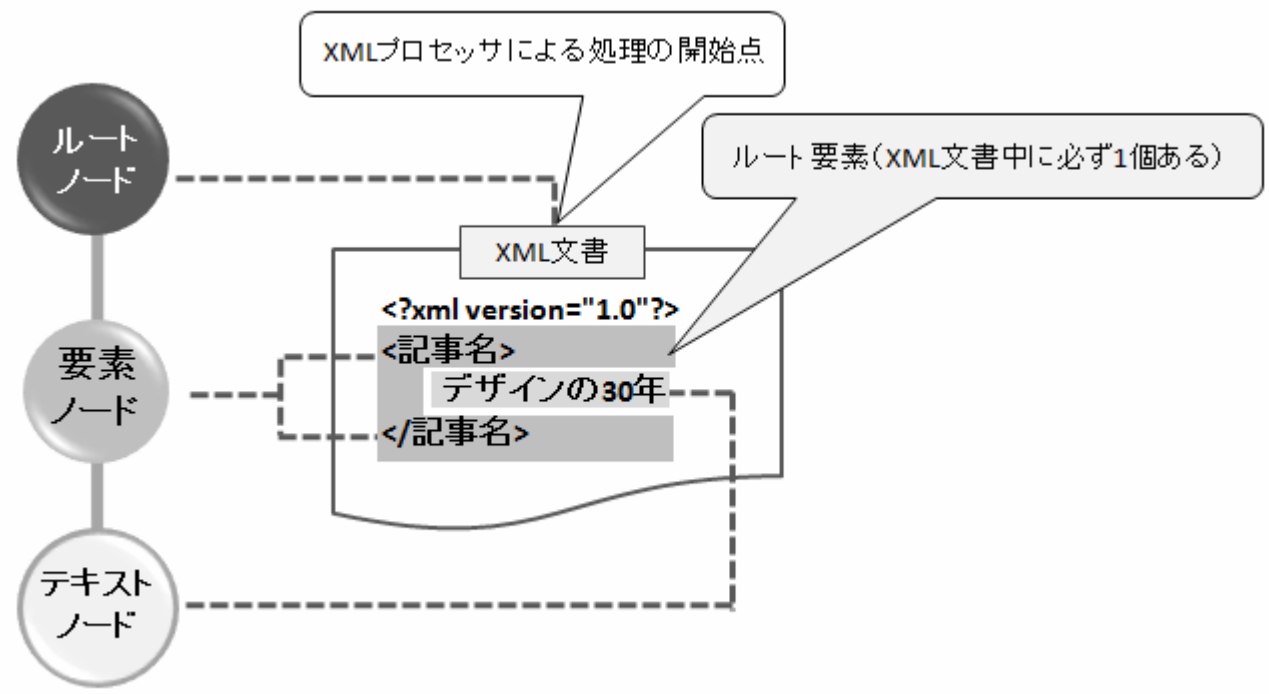
●図 2-2　妥当な XML 文書は，整形式 XML 文書の上に成り立つ



●図 2-3　整形式 XML 文書の基本的な構成



●図 2-4　ルートノードとルート要素



●表 2-1　XML の名前文字の一例

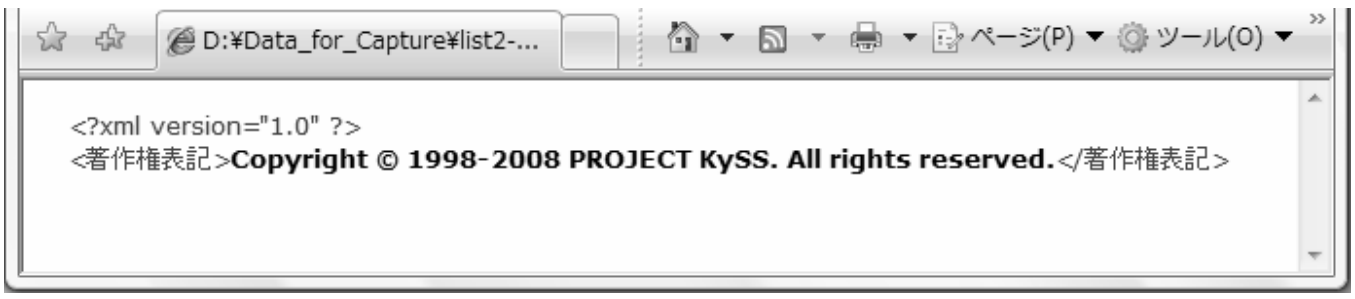
使ってもよい例	
〈型番〉	
〈ふりがな〉	
〈メニュー〉	
〈ISBN〉	
使ってはいけない例	
タグ	**理由**
〈2008年〉	数字から始まっている
〈first name〉	スペースがある
〈XmlRoot〉	予約語が使われている
〈ﾙﾋﾞ〉	半角カタカナである

●表 2-2　定義済み実体

記号	文字参照
lt	<
gt	>
amp	&
apos	'
quot	"

●図 2-5　定義済み実体を使用した結果。IE による表示。

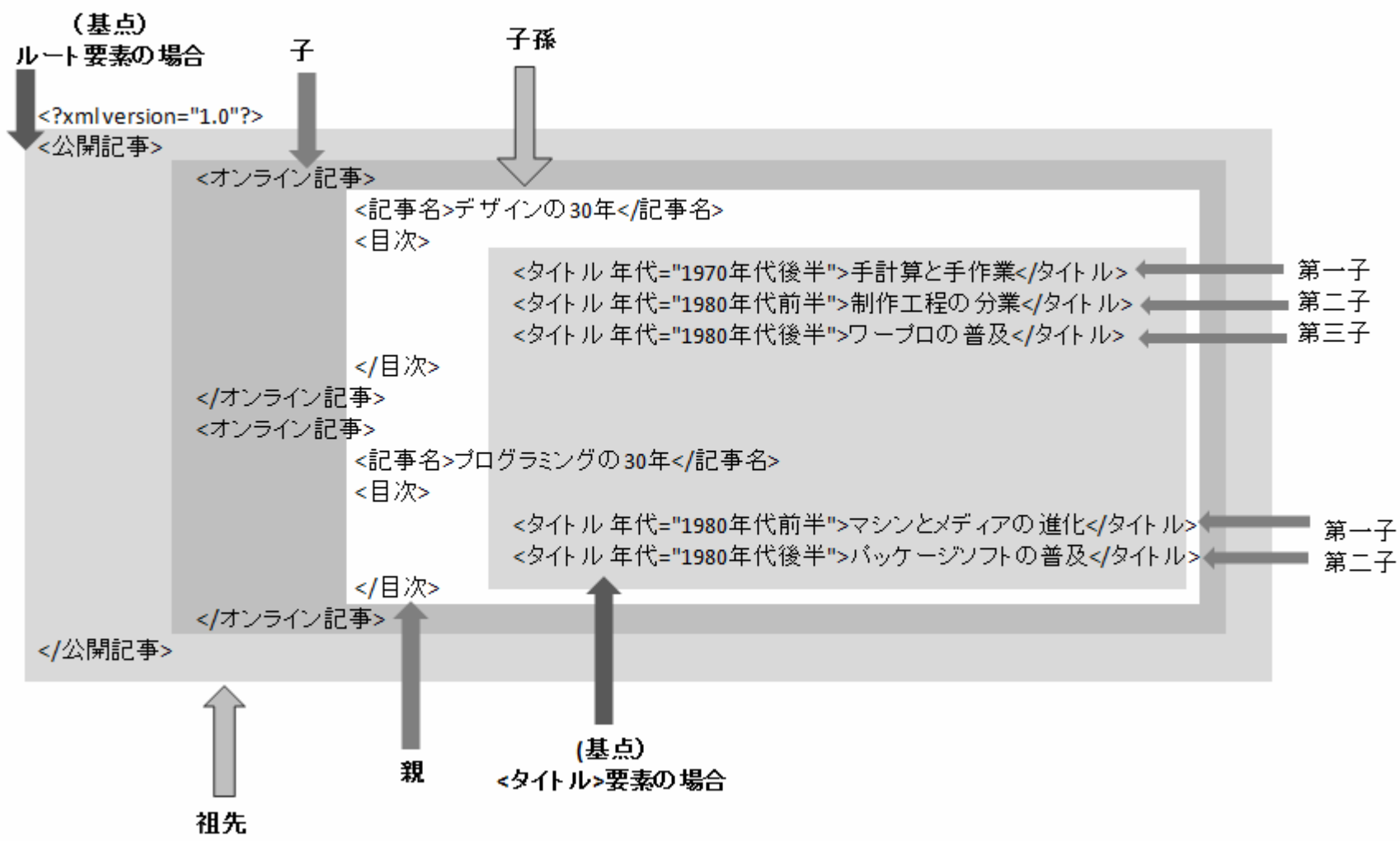
●図 2-6　文字参照により使用した結果



●表 2-3　式の一覧

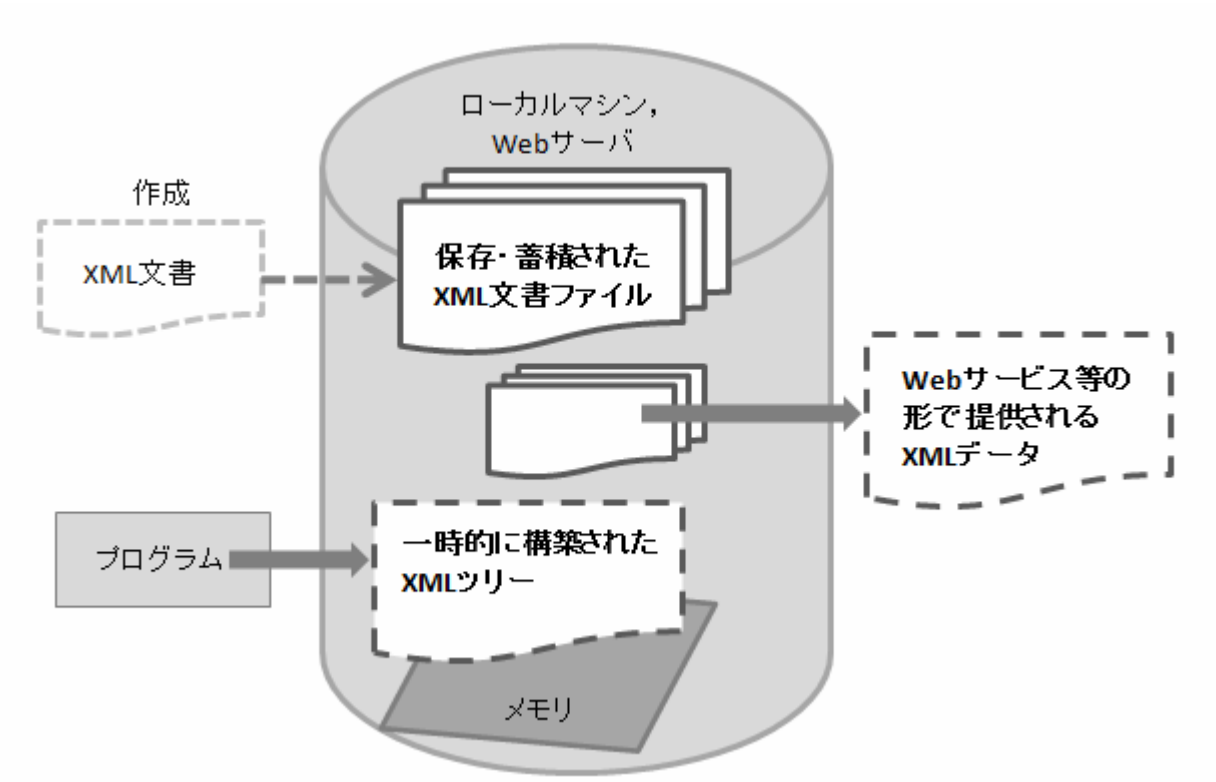
式	意味
A?	Aは出現(文書中に存在)してもよいし，出現しなくてもよい。オプション扱いである。
A｜B	AもしくはBのいずれか一方と合致する。｜はパイプ記号といい，orを表す。
A B	Aの次にBが出現する。演算子は選択より優先される。A B｜C D は (A B)｜(C D) と同じである。
A－B	AからBを除外した任意の文字列と合致する。差集合と考えればよい。
A+	Aの1回以上の繰返しと合致する。連結は選択より優先される。A+｜B+ は (A+)｜(B+) と同じである。
A*	Aの0回以上の繰返しと合致する。連結は選択より優先される。A*｜B* は (A*)｜(B*) と同じである。

●図 2-7　要素の最大階層が 4 階層の XML 文書



第 3 章の図表

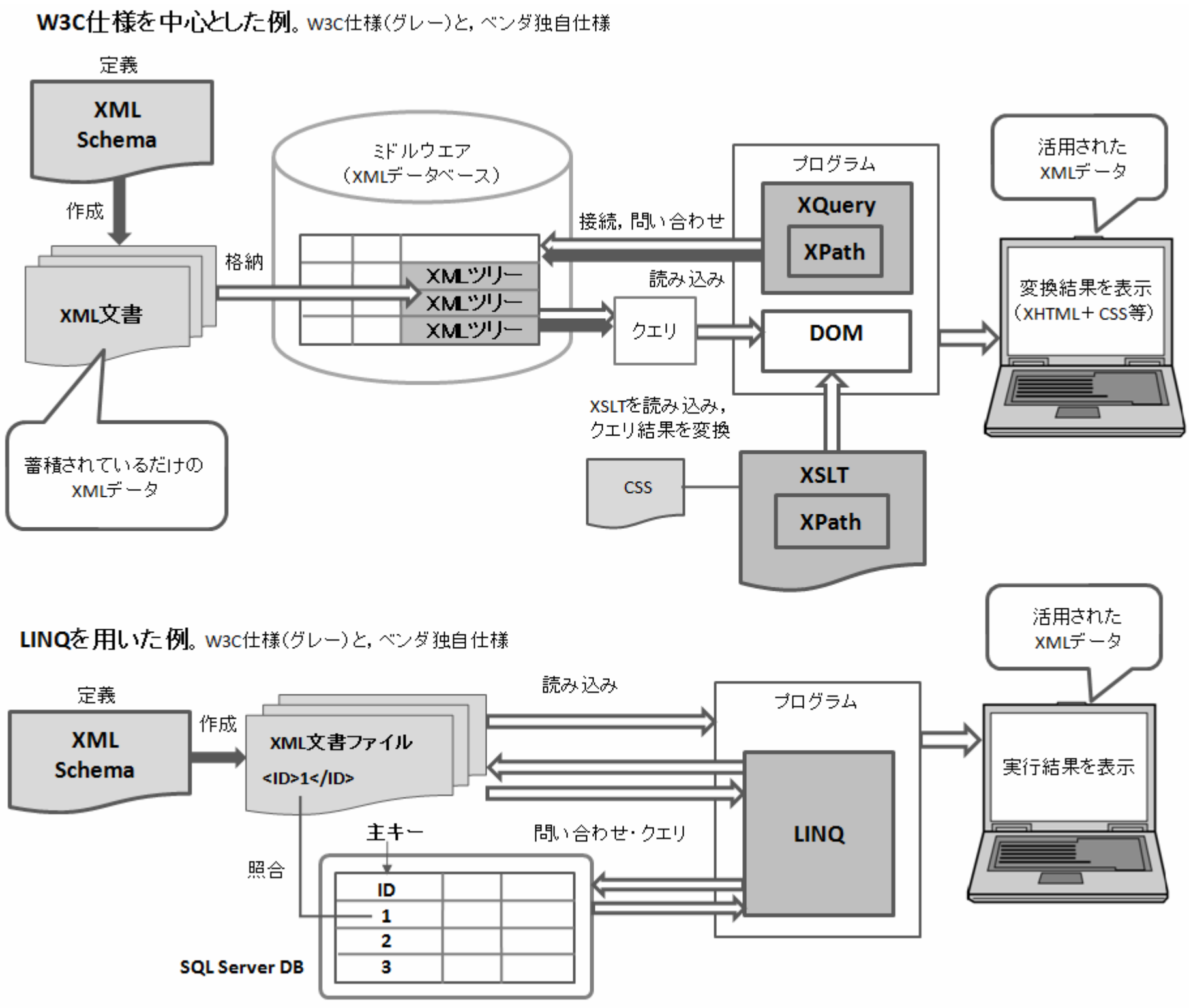
●図 3-1 さまざまな形で存在する XML データ



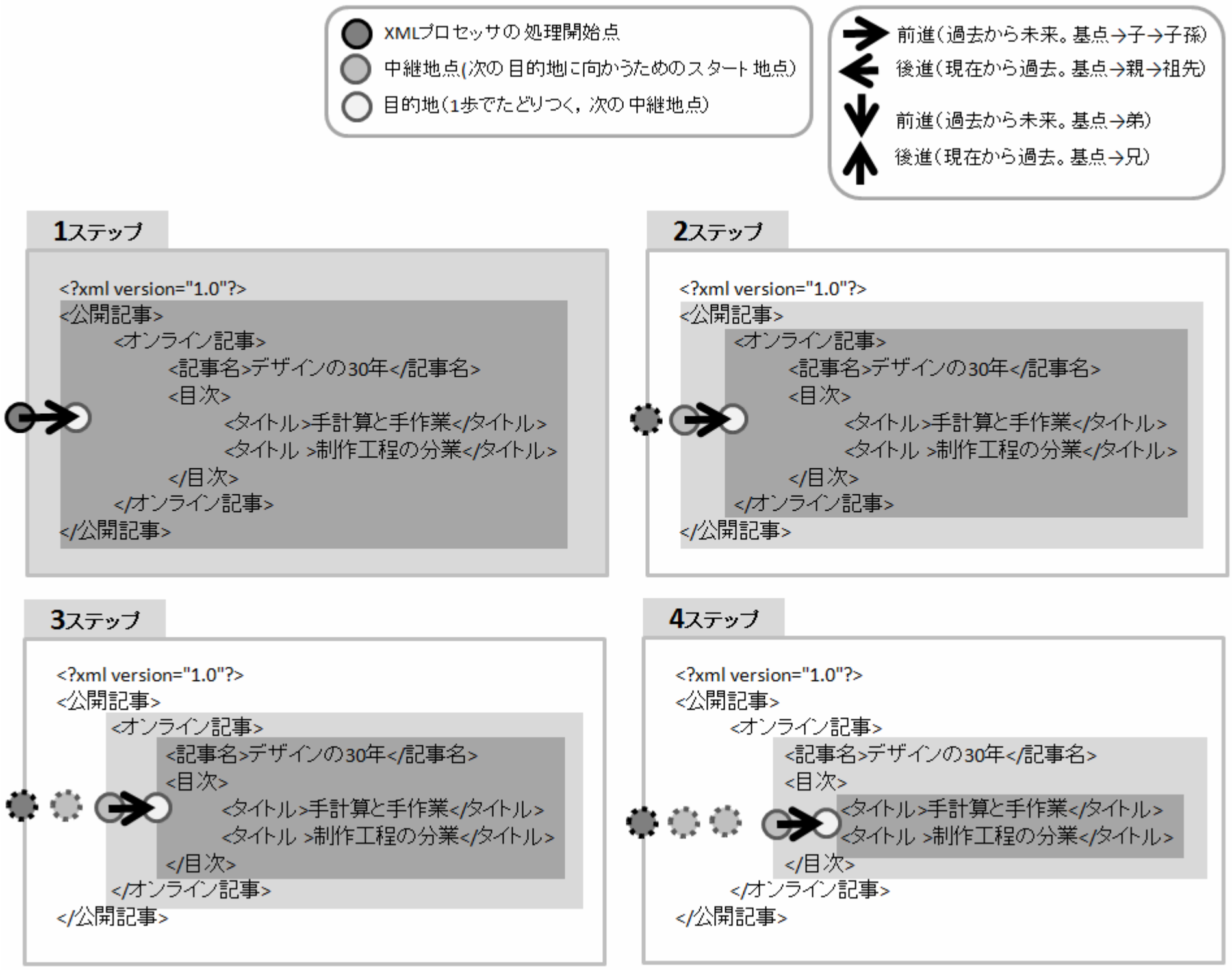
●表 3-1 XML および XML 周辺仕様の名称と機能

工程	仕様の名称	役割	策定機関
文書作成	**XML**（Extensible Markup Language）	XMLの基本仕様（第2章で解説）。	W3C
	Namespace in XML	同じ要素型や属性名を持つXML文書を利用／管理するための名前空間を指定する（第2章で解説）。	
データ処理	**XPath**（XML Path Language）	XML文書中の任意のノードをアドレス指定する仕様。XMLに関わる，あらゆる業務に必要（ノード間の関係については第2章で解説）。演算子と関数はXQueryとの共通仕様。	W3C
	XQuery（XML Query）	XML文書への問い合わせ言語。XMLデータベース機能を持つ複数のミドルウエアがサポートしており，中規模以上のXMLアプリケーション開発で多用される。XPathと組み合わせて使う。	
	DOM（Document Object Model）	XMLオブジェクトにアクセスして操作するためのAPI。プログラムコードの中で用いることにより，XML文書の構築，構造のナビゲート，要素や内容の追加・変更・削除を実装することができる。	
	XSLT（XSL Transformations）	XML文書を別のXML文書やHTMLに変換する言語。XML Webアプリケーション開発で多用される。XHTMLにも変換することができる。	
	LINQ（Language-Integrated Query）	.NET Framework3.5以降で動作するベンダ独自のクエリ技術。クエリの基本パターンは，RDB(LINQ to SQL)，XML(LINQ to XML)，Object(LINQ to Object)を問わず同一。XQuery，DOM，XSLTによる処理を代替え可能。	Microsoft
定義・検証	**XML Schema**	XML文書の構造を定義する。妥当なXML文書を検証する場合に必須。RelaxやDTDという選択肢もある。	W3C

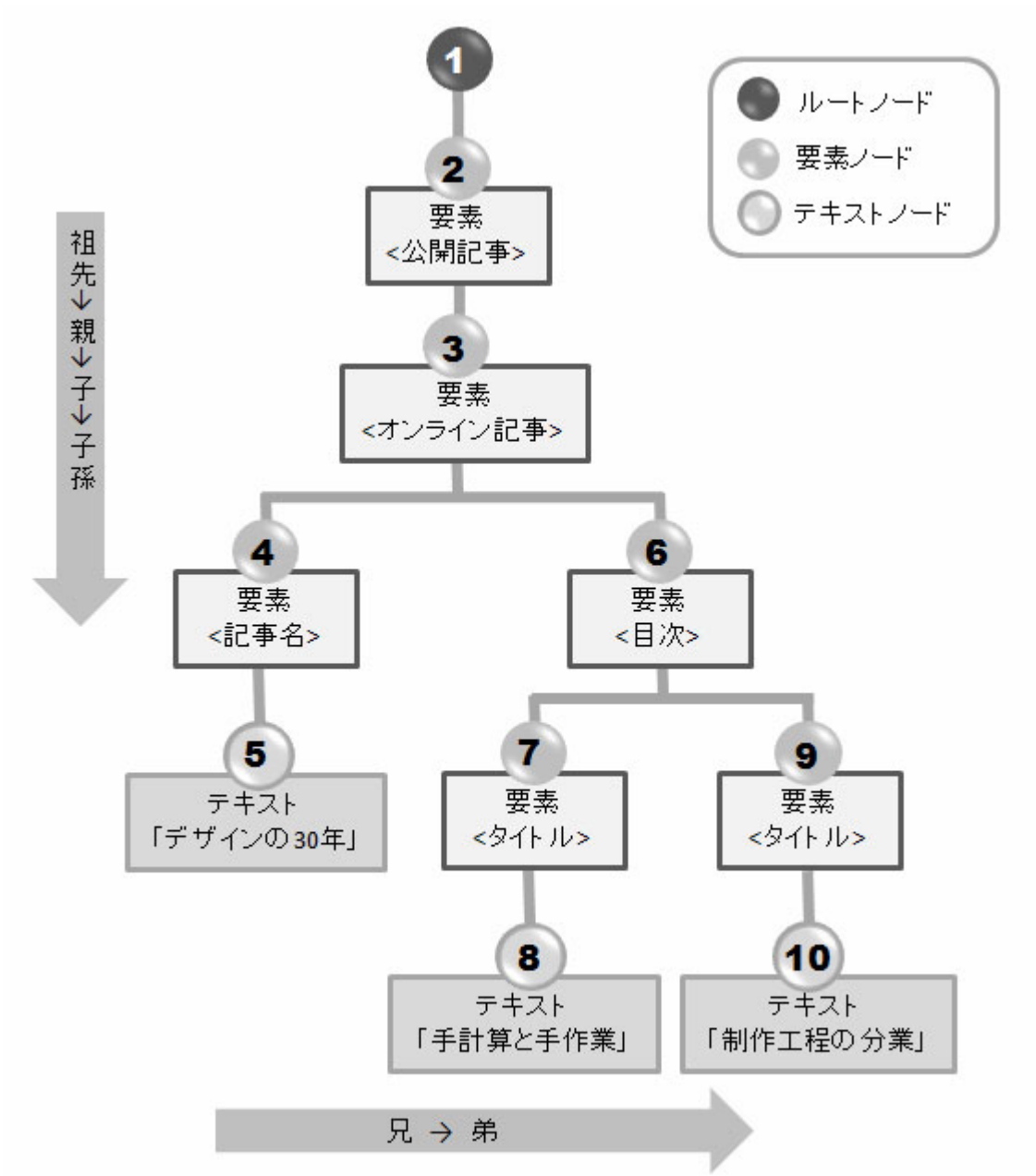
●図 3-2 XML 周辺仕様の併用方法の例



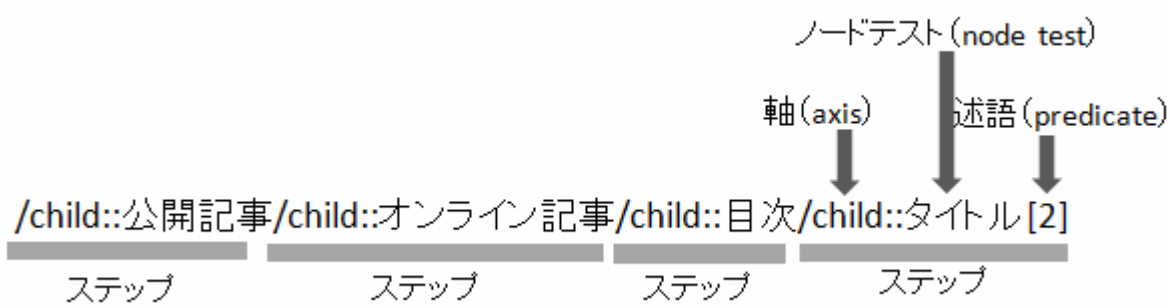
●図 3-3 XPath による処理対象の指定



●図 3-4 XML ツリーのノードの順序



●図 3-5 リスト 3-1 で，2 番目の<タイトル>を選択するパス式



●表 3-2 XPath の軸

前進軸(ForwardAxis)		後進軸(ReverseAxis)	
軸名	選択対象	軸名	選択対象
child	コンテキストノードの子を含む。	parent	コンテキストノードの親を返す。親がなければ空のシーケンスを返す。 属性ノードから見れば要素ノードは親にあたる。但し，要素ノードから見て属性ノードは子にはあたらない。
descendant	コンテキストノードの子要素をひっくるめた子孫を含む。これは，コンテキストノードの子と，子の子．．．というように続くものを指す。	ancestor	コンテキストノードの親要素をひっくるめた祖先を含む。これは，コンテキストノードの親と，親の親．．．というように続くものを指す。コンテキストノードがルートノードでない限り，ルートノードを含む。
descendant-or-self	コンテキストノードと，コンテキストノードの子孫群を含む。	ancestor-or-self	コンテキストノードと，コンテキストノードの祖先ノード群を含む。この軸は木をさかのぼるので，常に，ルートノードを含むことになる。
following-sibling	コンテキストノードの親の子供たちの中で，文書順ではコンテキストノードより後に出現する兄弟ノードを含む。コンテキストノードが属性あるいは名前空間ノードであれば，この軸は空である。DOM(後述)のnextSiblingと混同して憶えないように注意。	preceding-sibling	コンテキストノードの親の子供たちの中で，文書順ではコンテキストノードより前に出現する兄弟ノードを含む。コンテキストノードが属性あるいは名前空間ノードであれば，この軸は空である。DOM(後述)のpreviousSiblingと混同して憶えないように注意。
following	コンテキストノードと同じ文書内にあり，文書順でコンテキストノードの後に出現し，コンテキストノードの子孫を除くすべてのノード群を含む。	preceding	コンテキストノードと同じ文書内にあり，文書順でコンテキストノードの前に出現し，コンテキストノードの祖先を除くすべてのノード群を含む。
attribute	コンテキストノードの属性群を含む。コンテキストノードが要素でなければ空。		
self	コンテキストノード自身を含む。		
namespace	コンテキストノードの名前空間ノード群を含む。コンテキストノードが要素ノードでなければ空。この軸はXPath 2.0.では非推奨。XPath 1.0互換モード用の軸。		

●表 3-3 ノードテストの一例

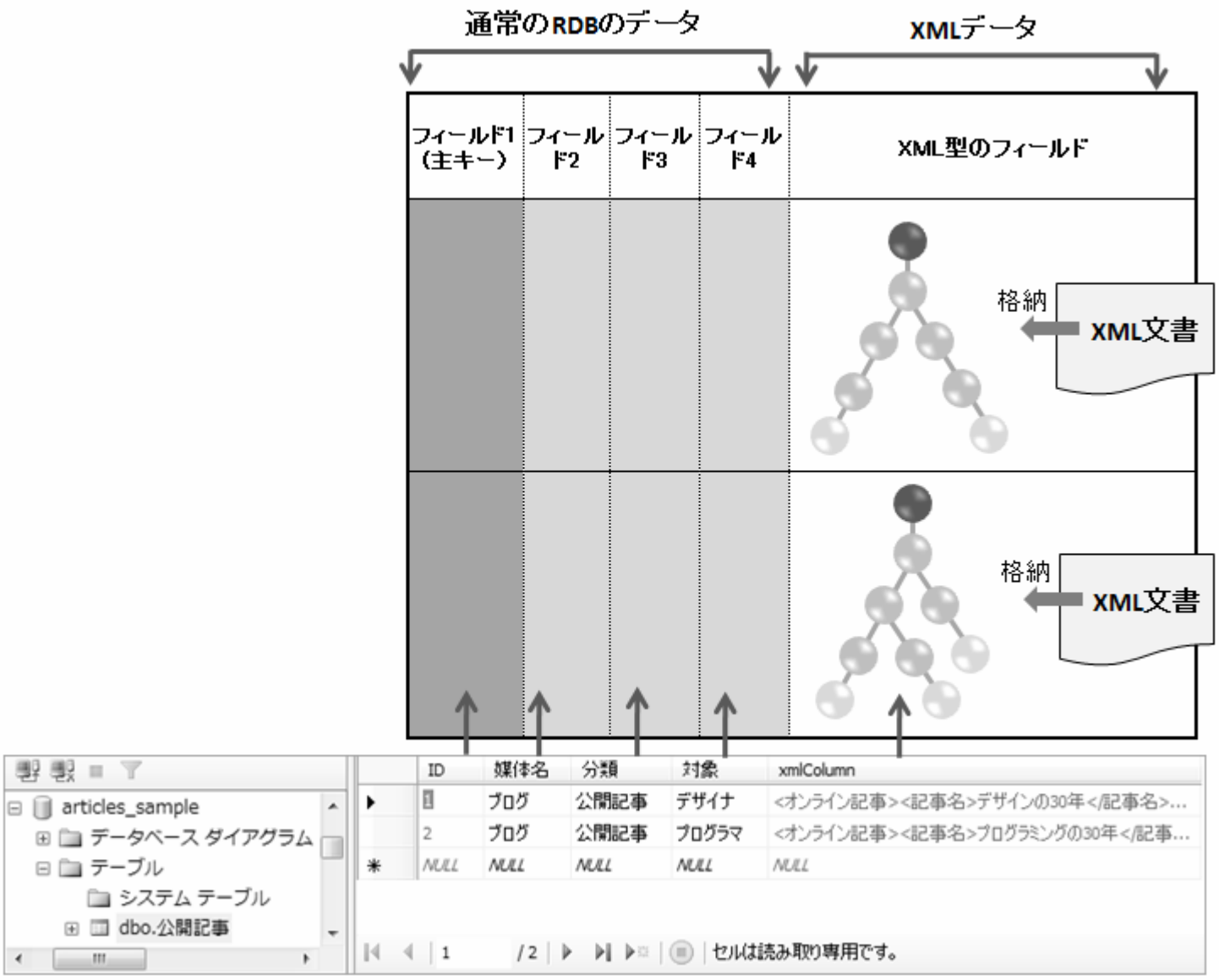
ノードテストの例	機能
node()	あらゆるノード型で真である。
text()	あらゆるテキストノードで真である。
comment()	あらゆるコメントノードで真である。
element()	あらゆる要素ノードで真である。
attribute()	あらゆる属性ノードで真である。
document-node()	あらゆる文書ノードで真である。

●表 3-4 XPath 式の記述例

選択対象	基本構文	省略構文	パス式の意味
子	child::要素C	要素C	コンテキストノードの子である 要素C を選択する
	child::*	*	コンテキストノードのすべての子要素を選択する
	child::node()	node()	コンテキストノードのすべての子を選択する。但し，属性は子ではないので返されない。
	child::要素C[fn:position() = 1]	要素C[1]	コンテキストノードの子の最初の要素Cを選択する
	child::要素C[fn:position() = fn:last()]	要素C[fn:last()]	コンテキストノードの子の最後の要素Cを選択する
	child::要素B[child::要素D = '概要']	要素B[要素D='概要']	<要素D>が「概要」という値を持つという条件に合致する，コンテキストノードの子である要素Bを選択する。
	child::要素B[attribute::属性A and attribute::属性B]	要素B[@属性A and @属性B]	"属性A"と"属性B"の両方を属性として持つ，コンテキストノードの子のすべての<要素B>を選択する。
親	parent::node()	..	コンテキストノードの親を選択する。コンテキストノードが属性ノードであれば，属性が付加されていても要素ノードを返す。
	parent::node()/child::要素B	../要素B	コンテキストノードの親の子の要素Bを選択する。
子孫	child::要素A/descendant-or-self::node()/child::要素D	要素A//要素D	子の要素Aのすべての子孫の要素Dを選択する。
祖先	ancestor::要素A		コンテキストノードの祖先のすべての要素Aを選択する。
兄弟	following-sibling::要素B[fn:position() = 1]	following-sibling::要素B[1]	コンテキストノードの後に続く兄弟である要素Bを選択する。
	preceding-sibling::要素B[fn:position() = 1]	preceding-sibling::要素B[1]	コンテキストノードの前に出現する兄弟である要素Bを選択する。
自分自身	self::要素C		コンテキストノードが要素Cであれば選択するが，そうでなければ空のシーケンスを返す
属性	attribute::属性A	@属性A	コンテキストノードの属性Aを選択する
	attribute::*	@*	コンテキストノードのすべての属性を選択する
テキスト	child::text()	text()	コンテキストノードのすべてのテキストノードを選択する

※ XPath1.0では，接頭辞fnを用いることなく関数を指定する。

●図 3-6 XQuery ではミドルウエアに格納した XML データを処理できる。



●表 3-5 FLWOR 式と各句の基本構文

FLWOR式	(For句あるいはLet句)+ Where句? OrderBy句? return 単一式
For句	for $ 変数名 as シーケンスの型? at $ 変数名? in 単一式 (, $ 変数名 as シーケンスの型? at $ 変数名? in 単一式)*
Let句	let $ 変数名 as シーケンスの型? := 単一式 (, $ 変数名 as シーケンスの型? := 単一式)*
Where句(オプション)	where 単一式
Return句	return 単一式
OrderBy句(オプション)	(order byあるいは stable order by) 単一式 ソート条件 (, 単一式 ソート条件)*

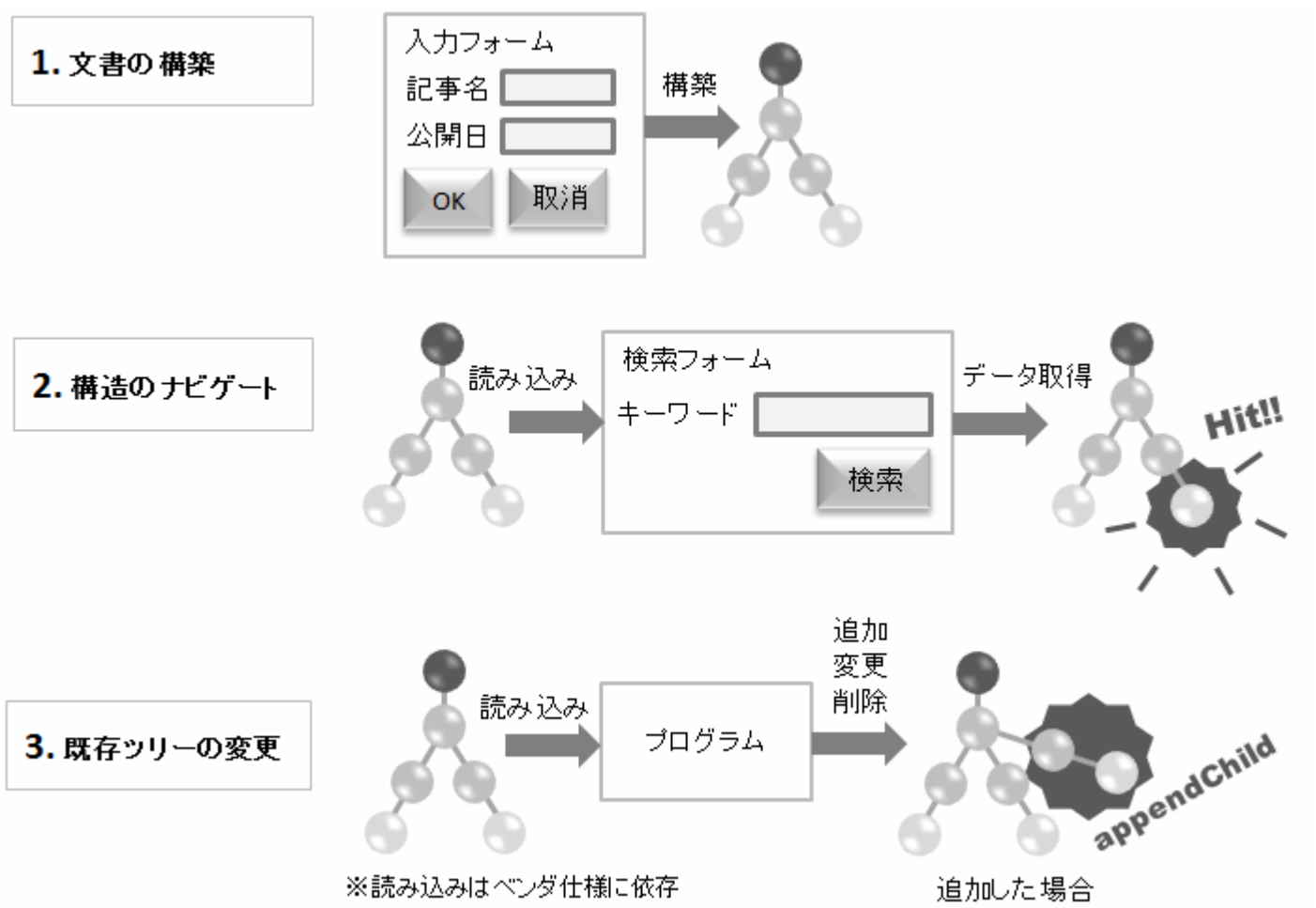
※第2章で触れたが，?，*，+は，出現回数を表す。? はオプション，* は0回以上，+ は1回以上である。
※For句で複数変数が使われる場合，たとえば，for $i in (a, b), $j in (p, r) では，(i,j) 成分という考え方でタプルストリームの生成を期待できる。
※For句とLet句の型宣言はオプションで，「2.5.3 SequenceType Syntax」の文法に従う。宣言した変数の型に，バインドされる値の型が合致しなければ，型エラーになる。
※OrderBy句のソート条件には，(ascending あるいは descending)? (empty (greatest あるいは least))? (collation リテラルURI)? を指定する。

●表 3-6 XQuery の関数と演算子の仕様（一部）

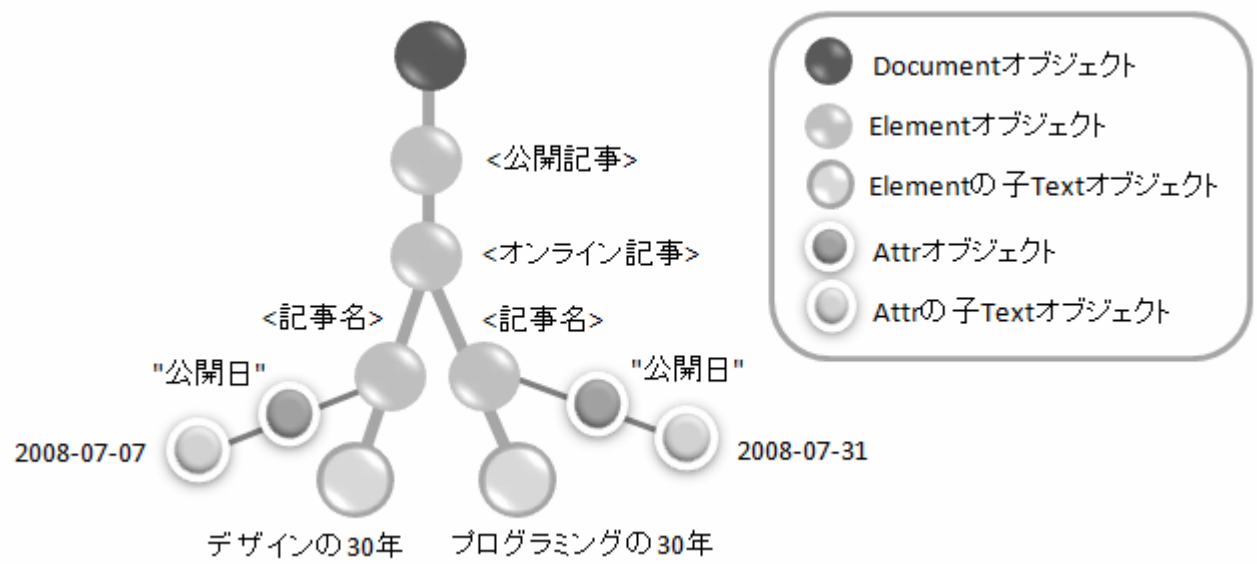
関数名，演算子名	主な機能	構文
fn:avg	平均値を算出できる。引数の値の合計を値の数で除算し，そのシーケンスの中での平均値を返す。 引数が空シーケンスの時は，空シーケンスを返す。 引数にxs:untypedAtomic型の値を含まれる場合は，xs:doubleにキャストされる。	fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?
fn:ceiling	引数の値よりも小さくはない関数の部分でない最も小さい値（負の無限大に近い）を返す。 引数の型がnumeric型（xs:float, xs:double, xs:decimal, xs:integer）の一つの場合，結果の型は，引数のに同じ。引数がnumeric型の一種であれば，numeric型をベースとするインスタンスになる。	fn:ceiling($arg as numeric?) as numeric?
fn:concat	，（カンマ）区切りで指定した文字列を連結できる。 2個以上のxs:anyAtomicTypeの引数を，xs:stringにキャストし，連結したxs:stringを返す。 引数が空であれば，0文字として扱う。 連結する文字によっては，正規化が必要になる。構文は，return normalize-unicode(concat($第1引数, $第2引数))	fn:concat($arg1 as xs:anyAtomicType?, $arg2 as xs:anyAtomicType?, …) as xs:string
fn:contains	第1引数の文字列内に，（文字の並び順はそのままで）少なくとも最低限一致する第2引数の文字列が含まれているかどうかを判別し，その結果をブール値で返す。部分一致検索に利用できる。	fn:contains($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean fn:contains($arg1 as xs:string?, $arg2 as xs:string?, $collation as xs:string) as xs:boolean
fn:count	引数の値の中のアイテム群のアイテム数を返す。空の場合は0を返す。ノードの個数を取得したい場合に利用できる。	fn:count($arg as item()*) as xs:integer
fn:data	引数に指定されたアイテム群のシーケンスを取得して，アトミック値のシーケンスを返す。属性値を取得したり，子要素すべての内容テキストをシーケンスとして取得するために利用できる。	fn:data($arg as item()*) as xs:anyAtomicType*
fn:distinct-values	引数の値から重複する集合の値を削除した結果のシーケンスを返す。 xs:untypedAtomic型の値はxs:string型であるとみなして処理される。重複しない要素の内容を取得したい場合に有効。	fn:distinct-values($arg as xs:anyAtomicType*) as xs:anyAtomicType* fn:distinct-values($arg as xs:anyAtomicType*, $collation as xs:string) as xs:anyAtomicType*
fn:empty	引数の値が空のシーケンスであればtrueを返し，そうでなければfalseを返す。要素や属性の有無を調べることができる。	fn:empty($arg as item()*) as xs:boolean
fn:false	値がfalseであるブール値を返す。 xs:boolean("0")に等しい。 fn:true() as xs:booleanでは，trueつまりxs:boolean("1")。	fn:false() as xs:boolean
fn:floor	一言で言えば，端数を切り捨てる。引数より大きくはない整数の最大値を返す。 Round関数が，引数に1.5を指定すれば2を返すのに対し，floor関数は1を返す。 引数と結果の型については，fn:ceilingに同じ。	fn:floor($arg as numeric?) as numeric?
fn:id	引数に指定された1個以上のIDREF値群の値に，合致するID値がある要素ノード群のシーケンスを返す。	fn:id($arg as xs:string*) as element()* fn:id($arg as xs:string*, $node as node()) as element()*
fn:last	動的コンテキストから，コンテキスト・サイズを返す。つまり，最後のノードを特定することができる。	fn:last() as xs:integer
fn:local-name	引数として指定した名前のローカル部分を返す。つまり，ローカル名を取得できる。 xs:stringの引数の名前のローカル部分を返す。引数が省略されている場合，デフォルトはコンテキスト・アイテム。	fn:local-name() as xs:string fn:local-name($arg as node()?) as xs:string
fn:local-name-from-QName	引数のローカル部分を表すxs:NCName（No Colon Name，コロンの付いていない名前）を返す。引数が空シーケンスの時は空シーケンスを返す。	fn:local-name-from-QName($arg as xs:QName?) as xs:NCName?
fn:max	最大値を取得するために使う。引数に与えられたシーケンスの中の最大値のアイテムを選択する。	fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType? fn:max($arg as xs:anyAtomicType*, $collation as string) as xs:anyAtomicType?
fn:min	最小値を取得するために使う。引数に与えられたシーケンスの中の最小値のアイテムを選択する。	fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType? fn:min($arg as xs:anyAtomicType*, $collation as string) as xs:anyAtomicType?
fn:namespace-uri-from-QName	xs:string型の引数の名前空間URIを返す。引数が空シーケンスであれば，空シーケンスを返す。引数に名前空間がなければ，0の文字列を返す。	fn:namespace-uri-from-QName($arg as xs:QName?) as xs:anyURI?
fn:not	第1引数の文字列内に，第2引数の文字列が含まれていないかどうかを判別結果を，ブール値で返す。つまり，fn:contains関数とは逆で，含まれていないものを抽出する。	fn:not($arg as item()*) as xs:boolean
fn:number	引数を示す値を返すか，引数が指定されていない場合は，xs:doubleに変換された，アトミック化の後のコンテキストアイテムを返す。	fn:number() as xs:double fn:number($arg as xs:anyAtomicType?) as xs:double

fn:position	動的コンテキストから，コンテキスト・ポジションを返す。ポジションを指定してノードを選択するために利用できる。	fn:position() as xs:integer
fn:string	文字列値として表わされる，アイテムの引数の値を返す。属性付き要素で，要素の内容テキストのみを取得することができる。	fn:string() as xs:string fn:string($arg as item()?) as xs:string
fn:string-length	文字列中の文字数の整数値を返す。引数の値が空のシーケンスの場合は0を返す。テキストノードの文字数を取得することができる。	fn:string-length() as xs:integer fn:string-length($arg as xs:string?) as xs:integer
fn:substring	文字列の始まりの位置を指定してデータを取得するために利用できる。第3引数で指定される文字数の，第2引数で指定した開始ロケーションのポジションから始まる，第1引数に指定した対象文字列の部分を返す。	fn:substring($sourceString as xs:string?, $startingLoc as xs:double) as xs:string fn:substring($sourceString as xs:string?, $startingLoc as xs:double, $length as xs:double) as xs:string
fn:sum	引数の値を加算した値を返す。	fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType fn:sum($arg as xs:anyAtomicType*, $zero as xs:anyAtomicType?) as xs:anyAtomicType?
fn:round	いわゆる「まるめ」。引数に最も近い整数の値を返す。 引数と結果の型については，fn:ceilingに同じ。	fn:round($arg as numeric?) as numeric?
op:union	重複するノード群を除いた2つの引数の和集合のノードのシーケンスを構築し，文書順で返す。	op:union($parameter1 as node()*, $parameter2 as node()*) as node()*
op:intersect	重複するノード群を除いた2つの引数の積集合のノードのシーケンスを構築し，文書順で返す。	op:intersect($parameter1 as node()*, $parameter2 as node()*) as node()*
op:except	第1引数にあって第2引数にない差集合のノードのシーケンスを構築し，文書順で返す。	op:except($parameter1 as node()*, $parameter2 as node()*) as node()*

●図 3-7 DOM の三大機能



●図 3-8 リスト 3-8 の DOM による表現



●表 3-7 構造モデル (DOM Level 1, Level2 「The DOM Structure Model」)

親のオブジェクト	子として持つことができるオブジェクト
Document	Element (ルート要素), ProcessingInstruction, Comment, DocumentType
DocumentFragment	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
EntityReference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Text, EntityReference
Entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
DocumentType	なし
ProcessingInstruction	なし
Comment	なし
Text	なし
CDATASection	なし
Notation	なし

●表 3-8 DOM Level1 コア，DOM Level 2 メソッド一覧 （「1.0」は Level1 のみ，「2.0」は Level2 での追加）

■基本インタフェース（Fundamental Interfaces ）

DOMImplementation			
特定のインスタンスに依存しない、独立した機能を提供する。			
メソッド	**createDocument**	文書要素の型で指定されたXMLのDocumentオブジェクトを生成する。	2.0
DOMImplementation.createDocument(namespaceURI, qualifiedName, doctype);			
パラメータ	namespaceURI	生成される文書要素のnamespaceURI。	
	qualifiedName	生成される文書要素の修飾名。	
	doctype of type	生成される文書の型あるいはnull。	
戻り値		新規Documentオブジェクト。	
メソッド	**createDocumentType**	空のDocumentTypeノードを生成する。	2.0
DOMImplementation.createDocumentType(qualifiedName, DOMString publicId, DOMString systemId);			
パラメータ	qualifiedName	生成される文書要素の修飾名。	
	publicId	外部サブセットの公開識別子。	
	systemId	外部サブセットのシステム識別子。	
戻り値		新規DocumentTypeオブジェクト。使われていない場合はnull。	
メソッド	**hasFeature**	指定する機能がDOMで実装されているかどうかを調べる	
DOMImplementation.hasFeature(feature,version);			
パラメータ	feature	テストする機能。Level1ではHTMLとXML（大文字小文字不問）	
	version	テストする機能のバージョン番号。	
戻り値		ブール値。実装されている場合true、実装されていない場合false。	
DocumentFragment			
新規生成された文書の断片を保持する。操作対象の親代わりとして移動させることによって、部分抽出、ノードの移動、子ノードの挿入などを実装する。Nodeからの継承。			
Document			
XML文書のツリーのルートを表す。文書中のデータにアクセスする基点となる。Nodeからの継承。			
属性	**doctype**	文書型宣言。これがない場合はnullを返す。	
	implementation	文書を処理するDOMImplementationオブジェクト。	
	documentElement	文書のルート要素の子ノードへの直接アクセスを許可する。	
メソッド	**createElement**	指定した型の要素を生成する。	
Document.createElement(tagName);			
パラメータ	tagName	インスタンス化する要素型。つまり要素名。	
戻り値		新規Elementオブジェクト。	
メソッド	**createElementNS**	修飾名と名前空間URIを指定して要素を生成する。	2.0
Document.createElementNS(namespaceURI,qualifiedName);			
パラメータ	namespaceURI	生成する要素の名前空間URI。	
	qualifiedName	インスタンス化される要素型の修飾名。	
戻り値		新規Elementオブジェクト。	
メソッド	**createDocumentFragment**	文書の断片のオブジェクトを生成する 。	
Document.createDocumentFragment();			
戻り値		新規DocumentFragmentオブジェクト。	
メソッド	**createTextNode**	指定した文字列からテキストノードを生成する。	
Document.createTextNode(data);			
パラメータ	data	テキストノードとなるデータ。	
戻り値		新規Textオブジェクト。	
メソッド	**createComment**	指定した文字列からコメントノードを生成する。	
Document.createComment(data);			
パラメータ	data	コメントノードとなるデータ。	
戻り値		新規Commentオブジェクト。	
メソッド	**createCDATASection**	指定した文字列からCDATASectionオブジェクトを生成する。	
Document.createCDATASection(data);			
パラメータ	data	CDATAセクションの内容となるデータ。	
戻り値		新規CDATASectionオブジェクト。	
メソッド	**createProcessingInstruction**	指定した名前とデータ文字列から処理命令ノードを生成する。	
Document.createProcessingInstruction(target, data);			
パラメータ	target	処理命令のターゲット部分。（<?の後続の文字列）	
	data	データ部分。（targetの後続の文字列）	
戻り値		新規ProcessingInstructionオブジェクト。	
メソッド	**createAttribute**	与えられた名前のAttrを生成する。Attrインスタンスは、setAttributeメソッドを使って要素にセットすることができる。	
Document.createAttribute(name);			
パラメータ	name	属性の名前。	
戻り値		新規Attrオブジェクト	
メソッド	**createAttributeNS**	修飾名と名前空間URIを指定して属性を生成する。	2.0
Document.createAttributeNS(namespaceURI,qualifiedName);			
パラメータ	namespaceURI	生成する属性の名前空間URI。	
	qualifiedName	生成する属性の修飾名。	
戻り値		新規Attrオブジェクト	
メソッド	**createEntityReference**	実体参照オブジェクトを生成する。	
Document.createEntityReference(name);			
パラメータ	name	参照する実体の名前。	
戻り値		新規EntityReferenceオブジェクト。	

メソッド	**getElementById**	elementIdに指定されたIDを持つElementを返す。	2.0
Document.getElementById(elementId);			
パラメータ	elementId	要素のユニークID名。	
戻り値		合致する要素。	
メソッド	**getElementsByTagName**	与えられたタグ名を持つ全ての要素のノードリストを返す。	
Document.getElementsByTagName(tagname);			
パラメータ	tagname	合致するタグの名前。*は全てのタグに合致する。	
戻り値		合致する全ての要素を含む新規NodeListオブジェクト。	
メソッド	**getElementsByTagNameNS**	修飾名と名前空間URIを指定して全ての要素のノードリストを返す。	2.0
Document.getElementsByTagNameNS(namespaceURI,localName);			
パラメータ	namespaceURI	合致する要素の名前空間URI。*は全ての名前空間に合致する。	
	localName	合致する要素のローカル名。*は全てのローカル名に合致する。	
戻り値		合致する全ての要素を含む新規NodeListオブジェクト。	
Node			
文書ツリーの中の単一のノードを表す。子を扱うためのメソッドを実装する。			
属性	**nodeName**	ノードの名前。	
	nodeValue	ノードの値。	
	nodeType	オブジェクトの型を表すコード。	
	parentNode	ノードの親。	
	childNodes	ノードの全ての子を含むノードリスト。	
	firstChild	ノードの最初の子。	
	lastChild	ノードの最後の子。	
	previousSibling	指定するノードの直前のノード。	
	nextSibling	指定するノードの後続のノード。	
	attributes	ノードの属性リストであるNamedNodeMapオブジェクト。	
	ownerDocument	【1.0】指定するノードに関連する文書オブジェクト。新規ノード群生成のために使われた文書オブジェクト。 【2.0】一部変更。	2.0
	localName	ノードの修飾名のローカル部分	
	namespaceURI	ノードの名前空間URI。	
	prefix	ノードの名前空間接頭辞。	
メソッド	**insertBefore**	既存の子ノードrefChild の前にnewChildノードを挿入する。refChildがnullの場合は、子のリストの最後にnewChildを挿入する。	
Node.insertBefore(newChild, refChild)			
パラメータ	newChild	挿入されるノード。	
	refChild	参照されるノード(新規ノード挿入位置の前にあるノード)。	
戻り値		挿入されたノード。	
メソッド	**replaceChild**	子のリストの中でoldChildをnewChildで置き換え、oldChildノードを返す。	
Node.replaceChild(newChild,oldChild);			
パラメータ	newChild	子のリストの中に置かれた新規ノード。	
	oldChild	リストの中で置き換えられるノード。	
戻り値		置き換えられたノード。	
メソッド	**removeChild**	子のリストからoldChildで示される子ノードを取り除き、これを返す。	
Node.removeChild(oldChild);			
パラメータ	oldChild	リムーブするノード(※注1参照)。	
戻り値		リムーブしたノード。	
メソッド	**appendChild**	ノードの子のリストの最後にnewChildを追加する。	
Node.appendChild(newChild);			
パラメータ	newChild	追加するノード。	
戻り値		追加されたノード。	
メソッド	**hasAttributes**	要素ノードが属性を持っているかどうかを調べる。	2.0
Node.hasAttributes();			
戻り値		属性を持っている場合はtrue、持っていない場合はfalse。	
メソッド	**hasChildNodes**	ノードが子を持っているかどうかを調べる。	
Node.hasChildNodes();			
戻り値		子を持っている場合はtrue、持っていない場合はfalse。	
メソッド	**cloneNode**	ノードの複製を返す。ノード群を複製するコンストラクタ。	
Node.cloneNode(deep);			
パラメータ	deep	trueの場合は、指定したノードの下にサブツリーを複製し、falseの場合は、ノード自身および(ノードが要素の場合は)その属性も複製する。	
戻り値		複製されたノード。	
メソッド	**isSupported**	指定したノードで機能がサポートされているかどうかを調べる。	2.0
Node.isSupported(feature,version);			
パラメータ	feature	テストする機能。Level1ではHTMLとXML(大文字小文字不問)	
	version	テストする機能のバージョン番号。	
戻り値		ブール値。実装されている場合true、実装されていない場合false。	
メソッド	**normalize**	【2.0】変更。全てのテキストノードを、隣接するテキストノードがないように、単一のノードに正規化する。	2.0
Node.normalize();			
NodeList			
順序のあるノード群の集合を表す。ノードリストのインデックス(添字)は0から始まる。※※注2参照。			
メソッド	**item**	指定するインデックス番号にあたるノード集合中のノードオブジェクトを返す。	
NodeList.item(index);			
パラメータ	index	ノード集合の中のインデックス番号。	
戻り値		ノードリストの中のインデックス番号を持つノード。	
属性	**length**	リストのノードの数。添字は0からlength-1までの範囲の値。	

NamedNodeMap			
名前でアクセスできるノードの集合を表す。NodeListからの継承はない。			
メソッド	**getNamedItem**	名前によって指定されたノードを取得する。	
NamedNodeMap.getNamedItem(name);			
パラメータ	name	取得するノードの名前。	
戻り値		指定された名前のノード。これがない場合はnull。	
メソッド	**getNamedItemNS**	ローカル名と名前空間URIによって指定されたノードを取得する。	2.0
NamedNodeMap.getNamedItemNS(namespaceURI, localName);			
パラメータ	namespaceURI	取得するノードの名前空間URI。	
	localName	取得するノードのローカル名。	
戻り値		ローカル名と名前空間URIによって指定されたノード。これがない場合はnull。	
メソッド	**setNamedItem**	nodeName属性を使っている属性をセットする。	
NamedNodeMap.setNamedItem(arg);			
パラメータ	arg	ノードマップにストアするノード。	
戻り値		既存ノードを新規ノードが同じ名前で置き換えた場合は既存のノード、それ以外の場合はnull。	
メソッド	**setNamedItemNS**	namespaceURIとlocalNameを使ってノードをセットする。	2.0
NamedNodeMap.setNamedItemNS(in Node arg);			
パラメータ	arg	ノードマップにストアするノード。	
戻り値		既存ノードを新規ノードが同じ名前で置き換えた場合は既存のノード、それ以外の場合はnull。	
メソッド	**removeNamedItem**	名前で指定されたノードをリムーブする。	
NamedNodeMap.removeNamedItem(name);			
パラメータ	name	リムーブするノードの名前。	
戻り値		マップからリムーブされたノード。指定する名前がない場合はnull。	
メソッド	**removeNamedItemNS**	ローカル名と名前空間URIによって指定されたノードをリムーブする。	2.0
NamedNodeMap.removeNamedItemNS(namespaceURI,localName);			
パラメータ	namespaceURI	リムーブするノードの名前空間URI。	
	localName	取得するノードのローカル名。	
戻り値		ローカル名と名前空間URIのあるマップからリムーブされたノード。これがない場合はnull。	
メソッド	**item**	マップ内のインデックス番号を持つアイテムを返す。	
NamedNodeMap.item(index);			
パラメータ	index	マップ内のインデックス番号。	
戻り値		指定するインデックス番号を持つマップ内のノード。	
属性	**length**	マップ内のノードの数。	
CharacterData			
DOM内の文字データにアクセスするための機能を提供する。Nodeからの継承。			
属性	**data**	処理対象となるノードの文字データ。	
	length	substringDataメソッドを使って利用可能な文字数。	
メソッド	**substringData**	ノードから一連のデータを抽出する。	
CharacterData.substringData(offset,count);			
パラメータ	offset	抽出する文字列の開始位置。	
	count	抽出する文字の数。	
戻り値		指定された文字列。	
メソッド	**appendData**	ノードの文字データの後に文字列を追加する。	
CharacterData.appendData(arg);			
パラメータ	arg	追加する文字列。	
メソッド	**insertData**	指定した位置に文字列を挿入する	
CharacterData.insertData(offset,arg);			
パラメータ	offset	挿入する文字の位置。	
	arg	挿入する文字列。	
メソッド	**deleteData**	ノードから指定する範囲の文字列をリムーブする。	
CharacterData.deleteData(offset,count);			
パラメータ	offset	文字をリムーブする位置。	
	count	削除する文字数。	
メソッド	**replaceData**	指定する文字で、指定するキャラクタオフセットで始まるキャラクタを置き換える。	
CharacterData.replaceData(offset,count,arg);			
パラメータ	offset	置換開始位置。	
	count	置換する文字数。	
	arg	置換する文字列。	
Attr			
要素オブジェクトの属性を表す。Nodeからの継承。			
属性	**name**	属性名を返す。	
	specified	元の文書の中で明らかに属性値がある場合はtrue，そうでなければfalse。	
	value	属性値を文字列として返す。	
	ownerElement	属性付き，あるいは属性が使われていない場合はnullの，Elementノード。	2.0

Element			
要素へのアクセスと操作を実装する。Nodeからの継承。			
属性	**tagName**	要素の名前。HTML DOMでは大文字が返されるが、XMLでは大文字小文字が区別されるので注意する。	
メソッド	**getAttribute**	指定した名前の属性値を取得する。	
Element.getAttribute(name);			
パラメータ	name	取得する属性の名前。	
戻り値		文字列としてのAttr値。属性値やデフォルト値がない場合は空の文字列。	
メソッド	**getAttributeNS**	ローカル名と名前空間URIで指定した属性値を取得する。	2.0
Element.getAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	取得する属性の名前空間URI。	
	localName	取得する属性のローカル名。	
戻り値		文字列としてのAttr値。属性値やデフォルト値がない場合は空の文字列。	
メソッド	**setAttribute**	新しい属性を追加する。	
Element.setAttribute(name,value);			
パラメータ	name	生成または変更する属性の名前。	
	value	セットする文字列値。	
メソッド	**setAttributeNS**	新規属性を追加する。要素に同じローカル名と名前空間URIを持つ属性がすでにある場合は，修飾名の接頭辞は変更され，valueパラメータでvalueが変更される。	2.0
Element.setAttributeNS(in DOMString namespaceURI,qualifiedName,value);			
	namespaceURI	生成後の属性の名前空間URI。	
パラメータ	localName	生成後の属性のローカル名。	
	value	セットする文字列値。	
メソッド	**removeAttribute**	指定する名前の属性をリムーブする。	
Element.removeAttribute(name);			
パラメータ	name	リムーブする属性の名前。	
メソッド	**removeAttributeNS**	ローカル名と名前空間URIで指定される属性をリムーブする。	2.0
Element.removeAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	リムーブする属性の名前空間URI。	
	localName	リムーブする属性のローカル名。	
パラメータ	name	リムーブする属性の名前。	
メソッド	**getAttributeNode**	名前によってAttrノードを取得する	
Element.getAttributeNode(name);			
パラメータ	name	取得する属性名。	
戻り値		指定する属性名を持つAttrノード。これがない場合はnull。	
メソッド	**getAttributeNodeNS**	ローカル名と名前空間URIで指定したAttrノードを取得する	2.0
Element.getAttributeNodeNS(namespaceURI,localName);			
パラメータ	namespaceURI	取得する属性の名前空間URI。	
	localName	取得する属性のローカル名。	
戻り値		指定する属性名を持つAttrノード。これがない場合はnull。	
メソッド	**setAttributeNode**	新しい属性を追加する。	
Element.setAttributeNode(newAttr);			
パラメータ	newAttr	属性リストに追加するAttrノード。	
戻り値		newAttr属性が既存の属性を同じ名前で置き換える場合は、既存のAttrノード。それ以外の場合はnull。	
メソッド	**setAttributeNodeNS**	新しい属性を追加する。	2.0
Element.setAttributeNodeNS(newAttr);			
パラメータ	newAttr	属性リストに追加するAttrノード。	
戻り値		newAttr属性が既存の属性を同じローカル名と名前空間URIで置き換える場合は、既存のAttrノード。それ以外の場合はnull。	
メソッド	**removeAttributeNode**	指定する属性をリムーブする。	
Element.removeAttributeNode(oldAttr);			
パラメータ	oldAttr	属性リストからリムーブするAttrノード。	
戻り値		リムーブされたAttrノード。	
メソッド	**getElementsByTagName**	与えられたタグ名の全ての子孫要素のノードリストを返す。	
Element.getElementsByTagName(name);			
パラメータ	name	合致するタグ名。*は全てのタグに合致する。	
戻り値		合致する要素ノードのリスト。	
メソッド	**getElementsByTagNameNS**	ローカル名と名前空間URIで与えられたタグ名の全ての子孫要素のノードリストを返す。	2.0
Element.getElementsByTagNameNS(namespaceURI,DOMString localName);			
パラメータ	namespaceURI	合致する要素の名前空間URI。*は全てに合致する。	
	localName	合致する要素のローカル名。	
戻り値		合致する要素すべてを含むノードのリスト。	
メソッド	**hasAttribute**	指定される名前を持つか，あるいはデフォルト値を持つ属性があるかどうかを調べる。	2.0
Element.hasAttributeNS(namespaceURI,localName);			
パラメータ	name	要素もしくはデフォルト値で指定される名前を持つ属性。	
戻り値		ブール値。要素で指定される名前を持つか，あるいはデフォルト値を持つ属性があればtrue，なければfalse。	
メソッド	**hasAttributeNS**	ローカル名と名前空間URIで指定される属性があるかどうかを調べる。	2.0
Element.hasAttributeNS(namespaceURI,localName);			
パラメータ	namespaceURI	検索される属性の名前空間URI。	
	localName	検索される属性のローカル名。	
戻り値		ブール値。ローカル名と名前空間URIで指定される属性が特定できるか，要素においてデフォルト値を持つ場合はtrue，それ以外はfalse。	
メソッド	**normalize**	全てのテキストノードを、隣接するテキストノードがないように、単一のノードに正規化する。	1.0

Text		
要素および属性のテキスト内容を表す。CharacterDataからの継承。		
メソッド	**splitText**	オフセット位置を指定して2つのテキストノードを兄弟ノードに区切る。
Text.splitText(offset);		
パラメータ	offset	0から始まるオフセット位置。
戻り値		新規テキストノード。
Comment		
コメントの内容、<!--と-->との間の全てのキャラクタを表す。CharacterDataからの継承。		

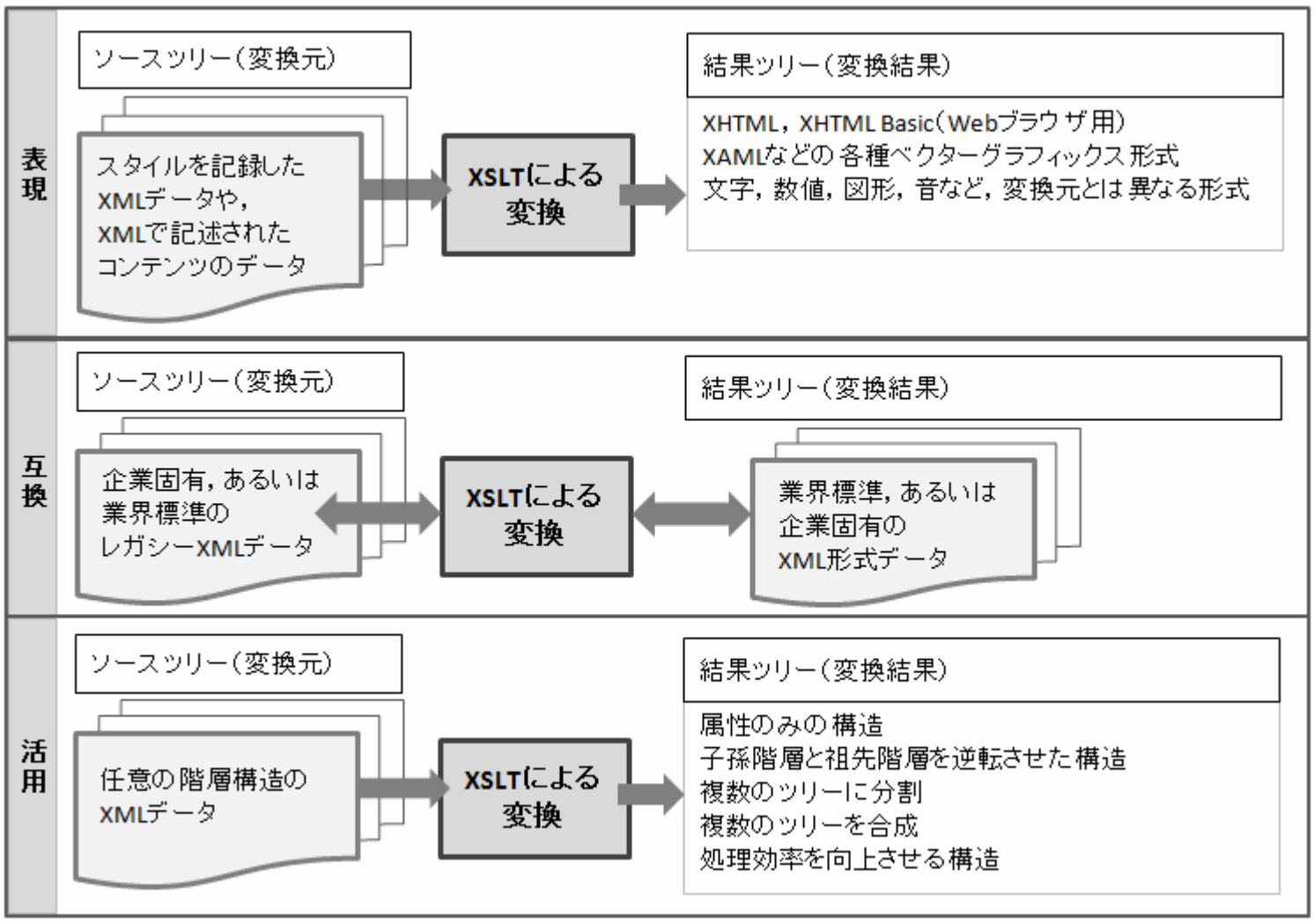
■**拡張インタフェース**（Extended Interfaces ）

CDATASection			
CDATAセクションを表す。（CDATAセクションは、マークアップと見なされるキャラクタを含むテキストブロックをエスケープするために使われる。CDATAセクションのデリミタは、]]>である。CDATAセクションは入れ子にできない。Textからの継承。			
DocumentType			
DTDを表す。Nodeからの継承。			
属性	**name**	DOCTYPEに続く，DTDの名前。	
	entities	内部実体、外部実体を含むNamedNodeMap。	
	notations	DTD内の記法宣言を含むNamedNodeMap。	
	internalSubset	文字列としての内部サブセット。	2.0
	publicId	外部サブセットの公開識別子。	
	systemId	外部サブセットのシステム識別子。	
Notation			
記法宣言を表す。Nodeからの継承。			
属性	**publicId**	公開識別子。指定されていない場合はnull。	
	systemId	システム識別子。指定されていない場合はnull。	
Entity			
パース対象実体、あるいはパース対象外実体を表す。Nodeからの継承。			
属性	**publicId**	公開識別子。指定されていない場合はnull。	
	systemId	システム識別子。指定されていない場合はnull。	
	notationName	パース対象外実体の記法の名前。パース対象実体についてはnull。	
EntityReference			
実体参照を表す。Nodeからの継承。			
ProcessingInstruction			
処理命令を表す。Nodeからの継承。			
属性	**target**	処理命令のターゲット。	
	data	処理命令の内容。	

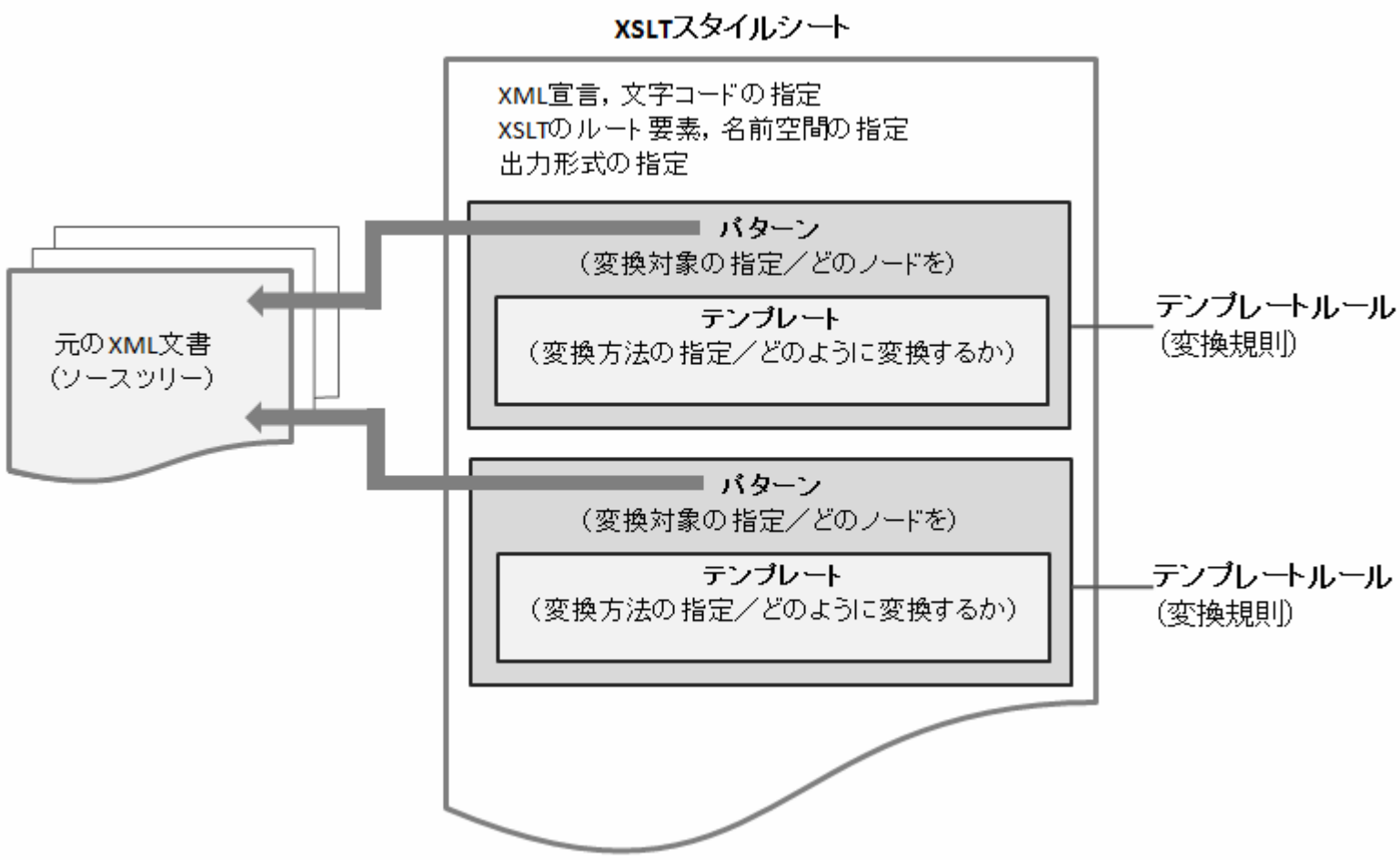
※注1
removeは構造モデルを変える時に使い、deleteは構造モデルの中にあるものを取り除く時に使う。deleteメソッドを使った場合は一度除去したものは返されないが、removeメソッドを使った場合は、返されることがある。

※※注2
DOMでは0から始まるインデックスを指定するが，XPathでは1から始まるポジションを指定する。プログラミング時に混同しないように注意が必要である。

●図 3-9 XSLT を用いて異なる構造の XML 文書に変換することができる。



●図 3-10 XSLT スタイルシートの構成



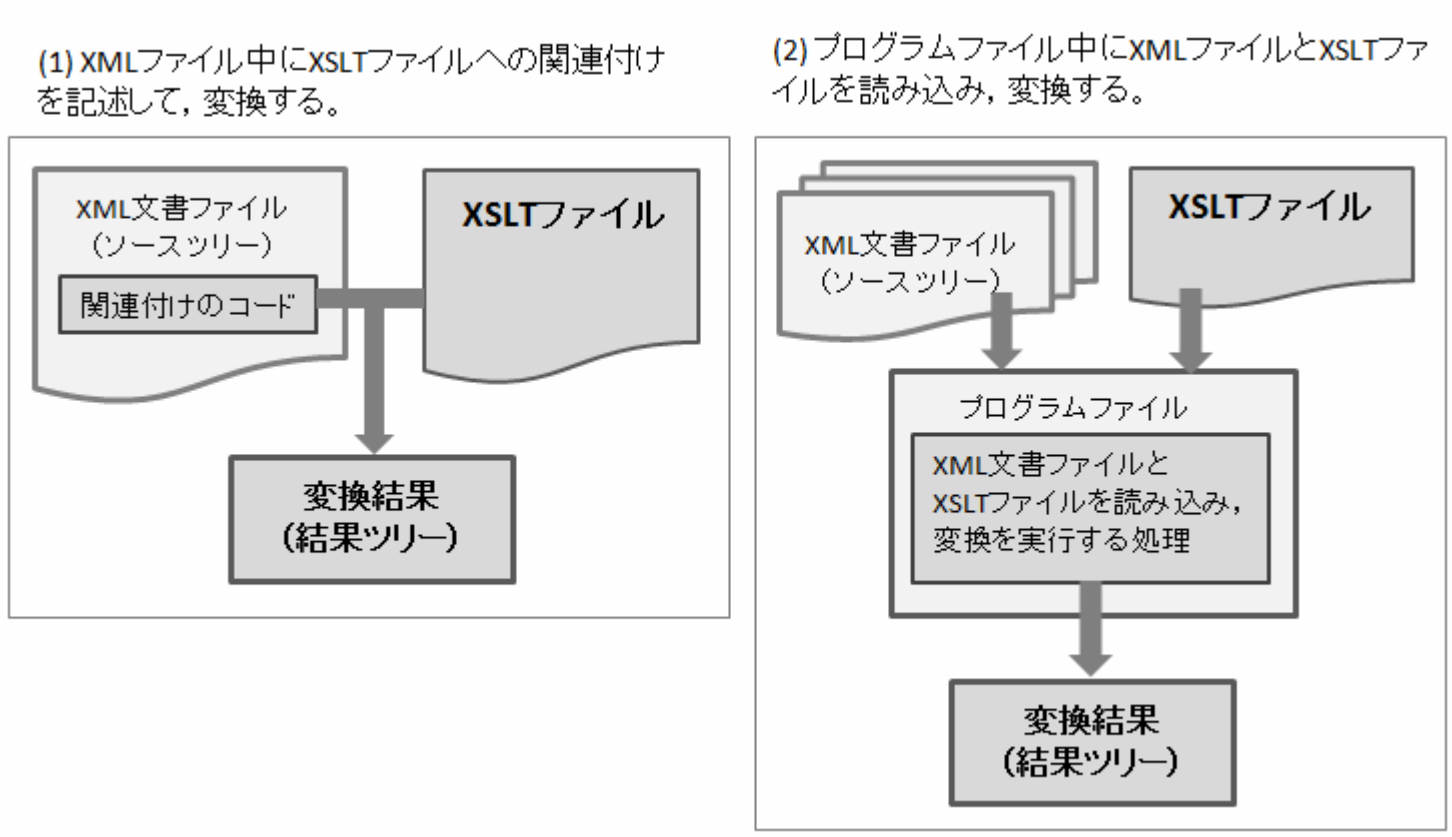
●表 3-9 XSLT1.0 の要素

スタイルシート要素		
xsl:stylesheet	他の全ての要素を含む，XSLTスタイルシートのルート要素。	<xsl:stylesheet id = id extension-element-prefixes = 接頭辞のリスト exclude-result-prefixes = 接頭辞のリスト version = バージョンを表す数字> <!-- テンプレートの内容: (xsl:import*, top-level-elements) --> </xsl:stylesheet>
xsl:transform	xsl:template要素とmsxsl:script要素を含むスタイルシートのルート要素。xsl:stylesheetと同義。	<xsl:transform id = id extension-element-prefixes = 接頭辞のリスト exclude-result-prefixes = 接頭辞のリスト version = バージョンを表す数字> <!-- テンプレートの内容: (xsl:import*, top-level-elements) --> </xsl:transform>
スタイルシートの結合		
xsl:include	別のXSLTスタイルシートをインクルードする。	<!-- Category: top-level-element --> <xsl:include href = URI参照 />
xsl:import	別のXSLTスタイルシートをインポートする。	<xsl:import href = URI参照 />
空白の扱い		
xsl:strip-space	空白を除去する。	<!-- Category: top-level-element --> <xsl:strip-space elements = 空白区切りのNameTestsのリスト />
xsl:preserve-space	空白を保持する。	<!-- Category: top-level-element --> <xsl:preserve-space elements = 空白区切りのNameTestsのリスト />
テンプレートルール		
xsl:template	テンプレートルールを定義する。	<!-- Category: top-level-element --> <xsl:template match = パターン name = テンプレートの名前 priority = 優先順位 mode = モード名> <!-- テンプレートの内容: (xsl:param*, template) --> </xsl:template>
xsl:apply-templates	テンプレートルールの適用。式で選択したノードを処理する。式が指定されていない場合は，テキストノードを含むカレントノードのすべての子を処理する。	<!-- Category: instruction --> <xsl:apply-templates select = ノード集合の式 mode = モード名> <!-- テンプレートの内容: (xsl:sort | xsl:with-param)* --> </xsl:apply-templates>
xsl:apply-imports	オーバーライドするテンプレートルールを呼び出す。	<!-- Category: instruction --> <xsl:apply-imports />
名前付けテンプレート		
xsl:call-template	名前でテンプレートを呼び出す。	<!-- Category: instruction --> <xsl:call-template name = テンプレートに付ける名前> <!-- 内容: xsl:with-param* --> </xsl:call-template>
結果ツリーの生成		
xsl:namespace-alias	別のネームスペースの接頭辞に置き換える。	<!-- Category: top-level-element --> <xsl:namespace-alias stylesheet-prefix = 変更元の接頭辞 | "#default" result-prefix = 変更後の接頭辞 | "#default" />
xsl:element	name属性に指定した名前の要素を生成する。	<!-- Category: instruction --> <xsl:element name = { 修飾名 } namespace = { URI参照 } use-attribute-sets = 属性セットの名前> <!-- テンプレートの内容: template --> </xsl:element>
xsl:attribute	name属性に指定した名前の属性を生成する。	<!-- Category: instruction --> <xsl:attribute name = { 属性名 } namespace = { URI参照 }> <!-- テンプレートの内容 --> </xsl:attribute>
xsl:attribute-set	名前を持つ属性の集合を定義する。	<!-- Category: top-level-element --> <xsl:attribute-set name = use-attribute-setsで使う属性集合の名前 use-attribute-sets = 空白で区切られた属性集合名のリスト> <!-- 内容: xsl:attribute* --> </xsl:attribute-set>
xsl:text	出力するテキストを生成する。	<!-- Category: instruction --> <xsl:text disable-output-escaping = "yes" | "no(デフォルト)"> <!-- テキスト化したいデータ: #PCDATA --> </xsl:text>
xsl:processing-instruction	出力する処理命令を生成する。	<!-- Category: instruction --> <xsl:processing-instruction name = { PIの名前 }> <!-- テンプレートの内容 --> </xsl:processing-instruction>
xsl:comment	出力するコメントを生成する。	<!-- Category: instruction --> <xsl:comment> <!-- コメントノードの文字列を示すテンプレート --> </xsl:comment>

xsl:copy	出力されるソースのカレントノードをコピーする。	<!-- Category: instruction --> <xsl:copy use-attribute-sets = 属性セット名> <!-- テンプレートの内容 --> </xsl:copy>
xsl:value-of	テキストを動的に生成する。	<!-- Category: instruction --> <xsl:value-of select = 文字列式 disable-output-escaping = "yes" | "no" />
ナンバリング		
xsl:number	結果ツリーに番号を挿入する。	<!-- Category: instruction --> <xsl:number level = "single" | "multiple" | "any" count = パターン from = パターン value = 数値式 format = { カウンタの表示形式 } lang = { 言語 } letter-value = { "alphabetic" | "traditional" } grouping-separator = { 桁区切り記号 } grouping-size = { 桁区切り桁数 } />
繰り返し		
xsl:for-each	select属性で指定した式で選択されるノードを繰り返し処理する。	<!-- Category: instruction --> <xsl:for-each select = ノード集合の式> <!-- テンプレートの内容: (xsl:sort*, template) --> </xsl:for-each>
条件分岐		
xsl:if	式を評価した結果が真ならテンプレートを処理する。	<!-- Category: instruction --> <xsl:if test = ブール式> <!-- テンプレートの内容 --> </xsl:if>
xsl:choose, xsl:when, xsl:otherwise	複数の条件を指定してテンプレートを処理する。 (xsl:when+, xsl:otherwise?)	<!-- Category: instruction --> <xsl:choose> <xsl:when test = ブール式> <!-- テンプレートの内容 --> </xsl:when> <xsl:otherwise> <!-- テンプレートの内容 --> </xsl:otherwise> </xsl:choose>
ソート		
xsl:sort	xsl:for-eachやxsl:apply-templatesで指定されたノードリストをソートする。	<xsl:sort select = 文字列式 lang = { 言語 } data-type = { "text" | "number" | qname-but-not-ncname } order = { "ascending" | "descending" } case-order = { "upper-first" | "lower-first" } />
変数とパラメータ		
xsl:variable	変数を宣言し，バインドする値を指定する。	<!-- Category: top-level-element --> <!-- Category: instruction --> <xsl:variable name = 変数名 select = 式> <!-- select属性を指定しない場合，処理されると変数値になるテンプレートの内容 --> </xsl:variable>
xsl:param	xsl:stylesheetまたはxsl:templateの中で使われる名前の付いたパラメータを宣言する。	<!-- Category: top-level-element --> <xsl:param name = パラメータ名 select = expression> <!-- select属性を指定しない場合，処理されるとパラメータ値になるテンプレートの内容 --> </xsl:param>
xsl:copy-of	結果ツリーにサブツリーと結果ツリーのフラグを挿入する。	<!-- Category: instruction --> <xsl:copy-of select = 式 />
xsl:with-param	テンプレートにパラメータを引き渡す。	<xsl:with-param name = パラメータ名 select = 式> <!-- テンプレートの内容 --> </xsl:with-param>
キー		
xsl:key	key関数で使う名前の付いたキーを宣言する。	<!-- Category: top-level-element --> <xsl:key name = キーの名前 match = パターン use = 式 />

数値の扱い		
xsl:decimal-format	format-numberで使われるフォーマットパターンの解釈の制御を指定する。	<!-- Category: top-level-element --> <xsl:decimal-format name = 名前 decimal-separator = 小数点記号，デフォルトは.(ピリオド) grouping-separator = グループ化の区切り記号，デフォルトは,(カンマ) infinity = 無限大を表す文字列，デフォルトはinfinity minus-sign = マイナス記号を表す文字，デフォルトはハイフンのマイナス文字 NaN = NaN型の値を表す文字列，デフォルトはNaN percent = パーセント記号を表す文字，デフォルトはパーセント文字(%) per-mille = 1000分の1記号，デフォルトはUnicodeのパーミル文字(#x2030) zero-digit = ゼロに使う文字，デフォルトはdigit文字のzero(0) digit = 数字の表記に使われる文字，デフォルトはナンバー記号文字の(#) pattern-separator = プラスとマイナスの区切り記号，デフォルトはセミコロン(;) />
メッセージ		
xsl:message	ダイアログボックスにテキストのメッセージを送信する。	<!-- Category: instruction --> <xsl:message terminate = "yes" | "no"> <!-- メッセージの内容 --> </xsl:message>
フォールバック		
xsl:fallback	解釈不可能な新しい要素に対して，代替となるテンプレートの内容を呼び出す。	<!-- Category: instruction --> <xsl:fallback> <!-- 代替になる処理可能なテンプレートの内容 --> </xsl:fallback>
出力		
xsl:output	結果ツリー構築のためのオプションを指定する。	<!-- Category: top-level-element --> <xsl:output method = "xml" | "html" | "text" | qname-but-not-ncname version = バージョン encoding = 文字コード omit-xml-declaration = "yes" | "no" standalone = "yes" | "no" doctype-public = 公開識別子 doctype-system = システム識別子 cdata-section-elements = CDATAセクション出力に使う要素名のリスト indent = "yes" | "no" media-type = MIMEコンテンツタイプ />

●図 3-11 XSLT スタイルシートによる変換を実行する方法



●表 3-10 XSLT の追加の関数

基本構文	機能
document(オブジェクト，ノー集合?)	メインのXML文書以外のXML文書へのアクセスを許可するノード集合関数
key(キーの名前，式の条件を表すオブジェクト)	xsl:key要素と組み合わせて，指定する動作を呼び出す，ノード集合関数
format-number(数値，フォーマットパターン文字列，文字列?)	2番目の引数のフォーマットパターン文字列と，3番目の引数のdecimal-format(あるいはデフォルト)で，1番目の引数を文字列に変換する文字列関数

●表 3-11 LINQ でサポートされている基本の句と機能

基本の句	機能
From	データを格納する変数名やデータの集合を指定
From句と組み合わせる句	**機能**
Select	投影(クエリの結果として選択される要素)
Where	フィルタリング(フィルタ条件)
Order By	ソート
Aggregate	クエリ結果を評価(関数併用など)
Join, Group Join	結合
Group By	グループ化
Let	式の評価結果を反復変数に格納
Distinct	重複するクエリを排除
Skip, Take	クエリ結果の一部を指定
Skip While, Take While	

●表 3-12 LINQ to XML の軸

LINQ to XML				XPath	
機能		基本構文	省略構文	種類	基本構文
コンテキストノードの子	前進軸	XContainer.Elements	.(1 個のピリオド)	前進軸	child
コンテキストノードの子孫	前進軸	XContainer.Discendants | XContainer.DiscendantNodes	...(3個のピリオド)		descendant
コンテキストノードと，コンテキストノードの子孫	前進軸	XElement.DiscendantAndSelf | XElement.DiscendantNodesAndSelf			descendant-or-self
コンテキストノードに続くすべての兄弟ノード	前進軸	XNode.ElementsAfterSelf | XNode.NodesAfterSelf			following-sibling
文書順でコンテキストノードより後にあり，子孫・属性・名前空間を除くすべてのノード	なし				following
コンテキストノードの属性群	前進軸	XElement.Attribute | XElement.Attributes	@		attribute
コンテキストノード自身	前進軸	XContainer.Nodes			self
コンテキストノードの名前空間	なし				namespace
コンテキストノードの親	前進軸	XObject.Parent	..(ダブルピリオド)	後進軸	parent
コンテキストノードの祖先	後進軸	XNode.Ancestors			ancestor
コンテキストノードと，コンテキストノードの祖先	後進軸	XElement.AncestorsAndSelf			ancestor-or-self
コンテキストノードの前にあるすべての兄弟ノード	前進軸	XNode.ElementsBeforeSelf | XNode.NodesBeforeSelf			preceding-sibling
文書順でコンテキストノードより前にあり，祖先・属性・名前空間を除くすべてのノード	なし				preceding

●図 3-12 リスト 3-35 の LINQ クエリの実行結果

```
<公開記事>
  <オンライン記事>
    <記事名>デザインの30年</記事名>
    <目次>
      <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
      <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
      <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
      <タイトル 年代="1990年代前半">DTPの普及</タイトル>
      <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
      <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
    </目次>
  </オンライン記事>
</公開記事>

<オンライン記事>
  <記事名>デザインの30年</記事名>
  <目次>
    <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
    <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
    <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
    <タイトル 年代="1990年代前半">DTPの普及</タイトル>
    <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
    <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
  </目次>
</オンライン記事>

<記事名>デザインの30年</記事名>

デザインの30年

<目次>
  <タイトル 年代="1970年代後半">手計算と手作業</タイトル>
  <タイトル 年代="1980年代前半">制作工程の分業</タイトル>
  <タイトル 年代="1980年代後半">ワープロの普及</タイトル>
  <タイトル 年代="1990年代前半">DTPの普及</タイトル>
  <タイトル 年代="1990年代後半">Webサイトの連携</タイトル>
  <タイトル 年代="2000年代前半">情報デザインの始まり</タイトル>
</目次>

<タイトル 年代="1970年代後半">手計算と手作業</タイトル>

手計算と手作業

<タイトル 年代="1980年代前半">制作工程の分業</タイトル>

制作工程の分業

<タイトル 年代="1980年代後半">ワープロの普及</タイトル>

ワープロの普及

<タイトル 年代="1990年代前半">DTPの普及</タイトル>

DTPの普及

<タイトル 年代="1990年代後半">Webサイトの連携</タイトル>

Webサイトの連携
```

<公開記事>要素ノード

<オンライン記事>要素ノード

<記事名>要素ノード

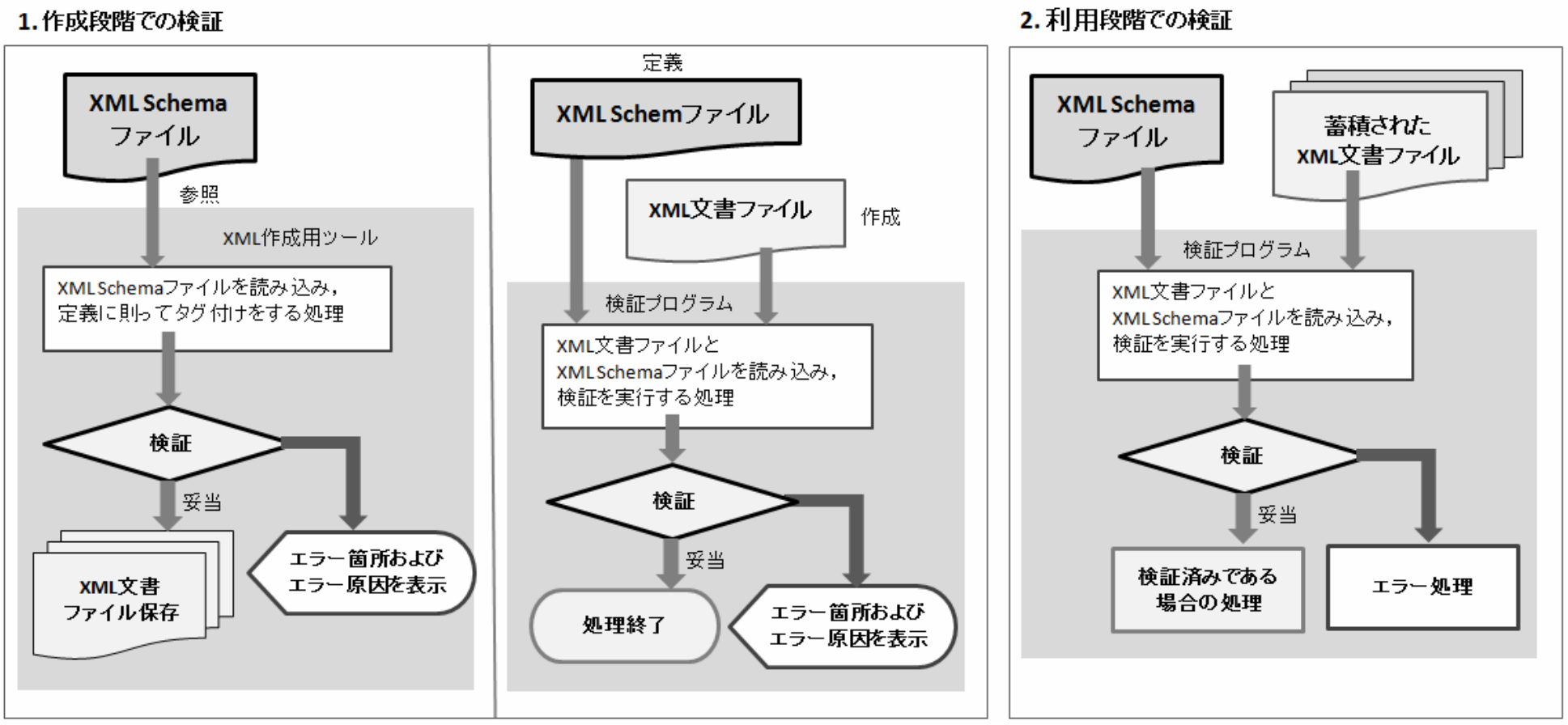
<記事名>要素内容のテキストノード

<目次>要素ノード

<タイトル>要素ノード

<タイトル>要素内容のテキストノード

●図 3-13 XML Schema 文書は，二重の検証機構となる。

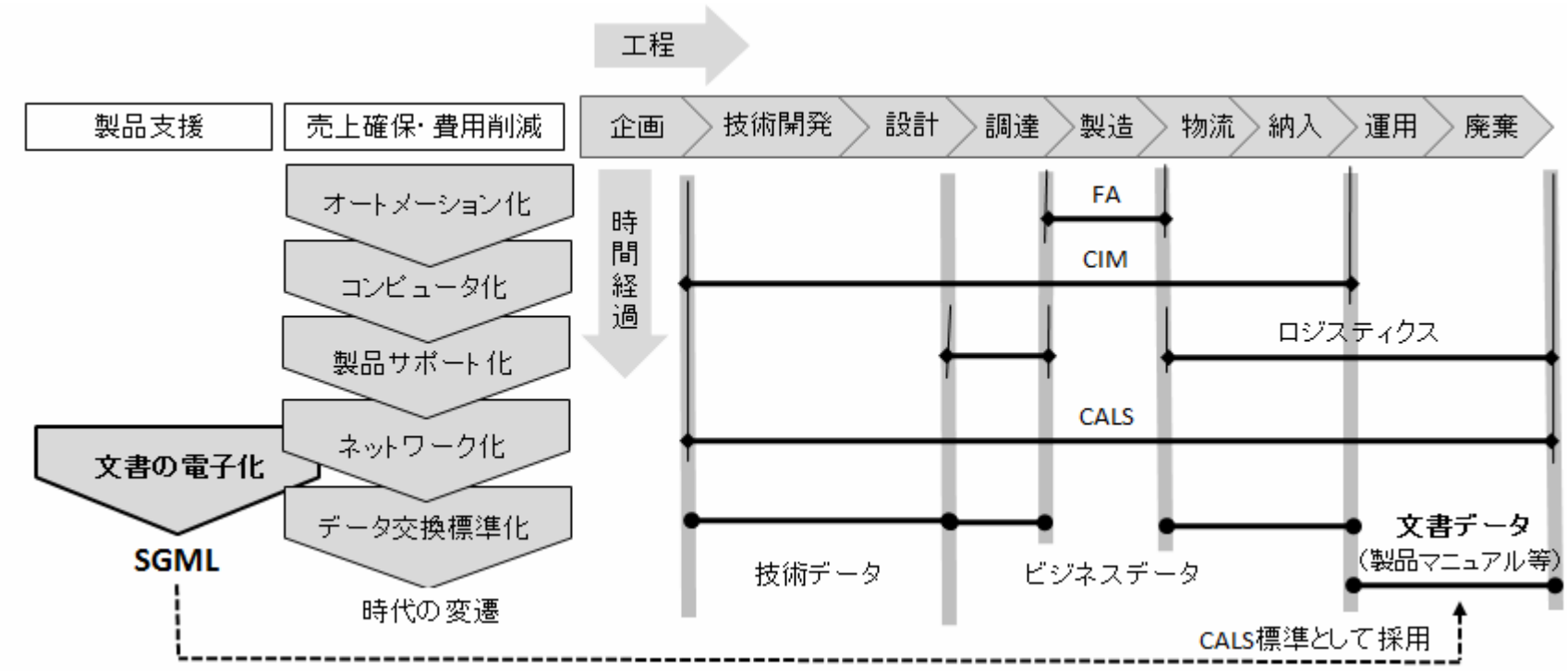


●表 3-13 単純型定義と複合型定義

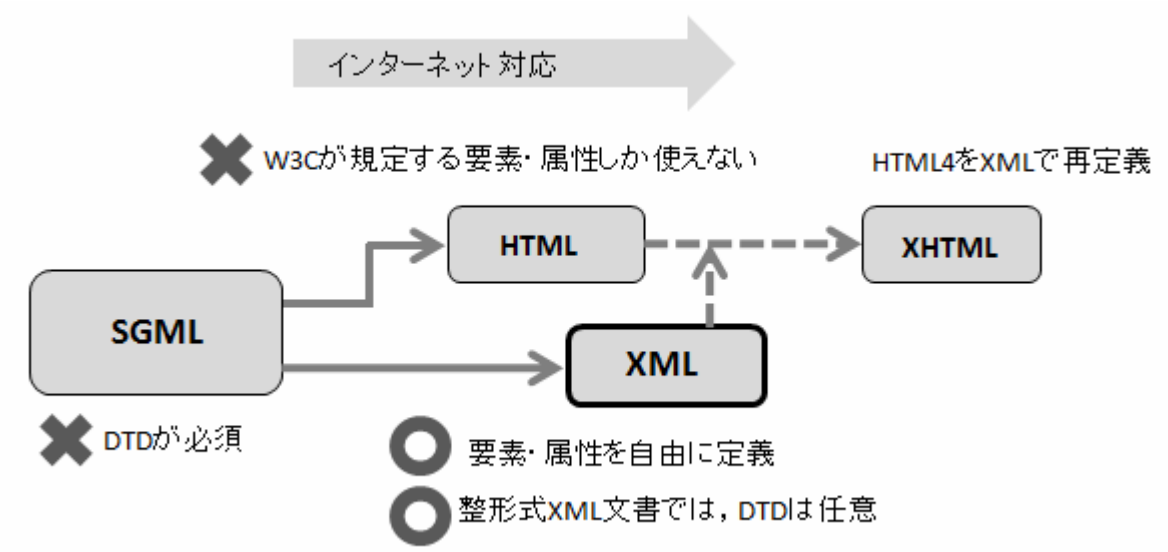
<table>
<tr><th></th><th></th><th colspan="2">単純型</th><th>複合型</th></tr>
<tr><td rowspan="3">要素</td><td>匿名の型定義</td><td colspan="2"><xsd:element name="要素名">
<xsd:simpleType>
データ型の派生の定義
</xsd:simpleType>
<xsd:element></td><td><xsd:element name="要素名">
<xsd:complexType>
複合型定義の内容
</xsd:complexType>
<xsd:element></td></tr>
<tr><td rowspan="2">名前付き型定義</td><td>宣言</td><td><xsd:element name="要素名" type="型定義の名前"/></td><td><xsd:element name="要素名" type="型定義の修飾名"/></td></tr>
<tr><td>型定義</td><td><xsd:simpleType name="単純型定義の名前">
データ型の派生の定義
</xsd:simpleType></td><td><xsd:complexType name="複合型定義の名前">
複合型定義の内容
</xsd:complexType></td></tr>
<tr><td rowspan="3">属性</td><td>匿名の型定義</td><td colspan="2"><xsd:attribute name="属性名">
<xsd:simpleType>
データ型の派生の定義
</xsd:simpleType>
<xsd:attribute></td><td></td></tr>
<tr><td rowspan="2">名前付き型定義</td><td>宣言</td><td><xsd:attribute name="属性名" type="型定義の名前"/></td><td></td></tr>
<tr><td>型定義</td><td><xsd:simpleType name="単純型定義の名前">
データ型の派生の定義
</xsd:simpleType></td><td></td></tr>
</table>

第 4 章の図表

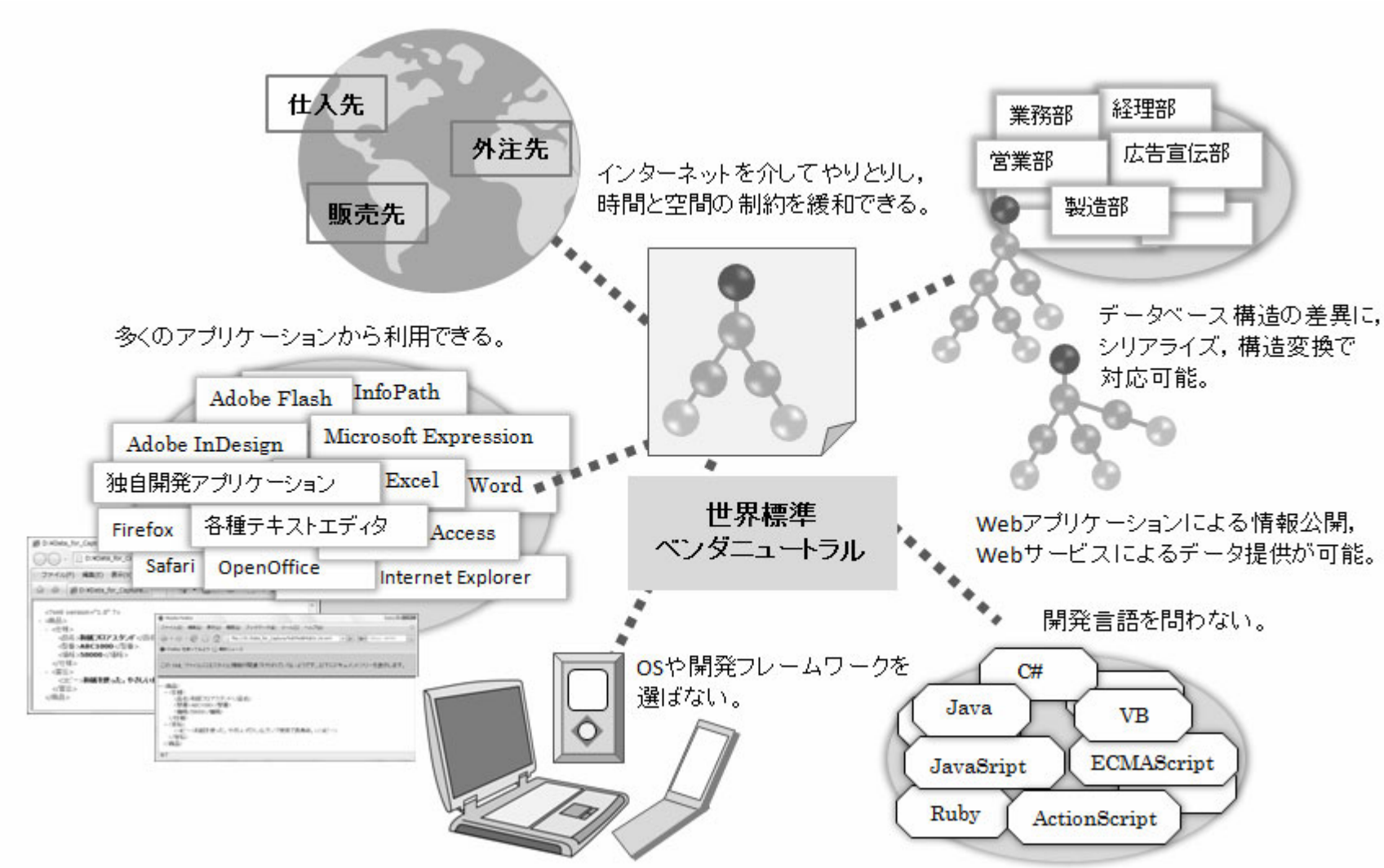
●図 4-1 XML 登場までの背景。ロジスティクス，CALS，SGML



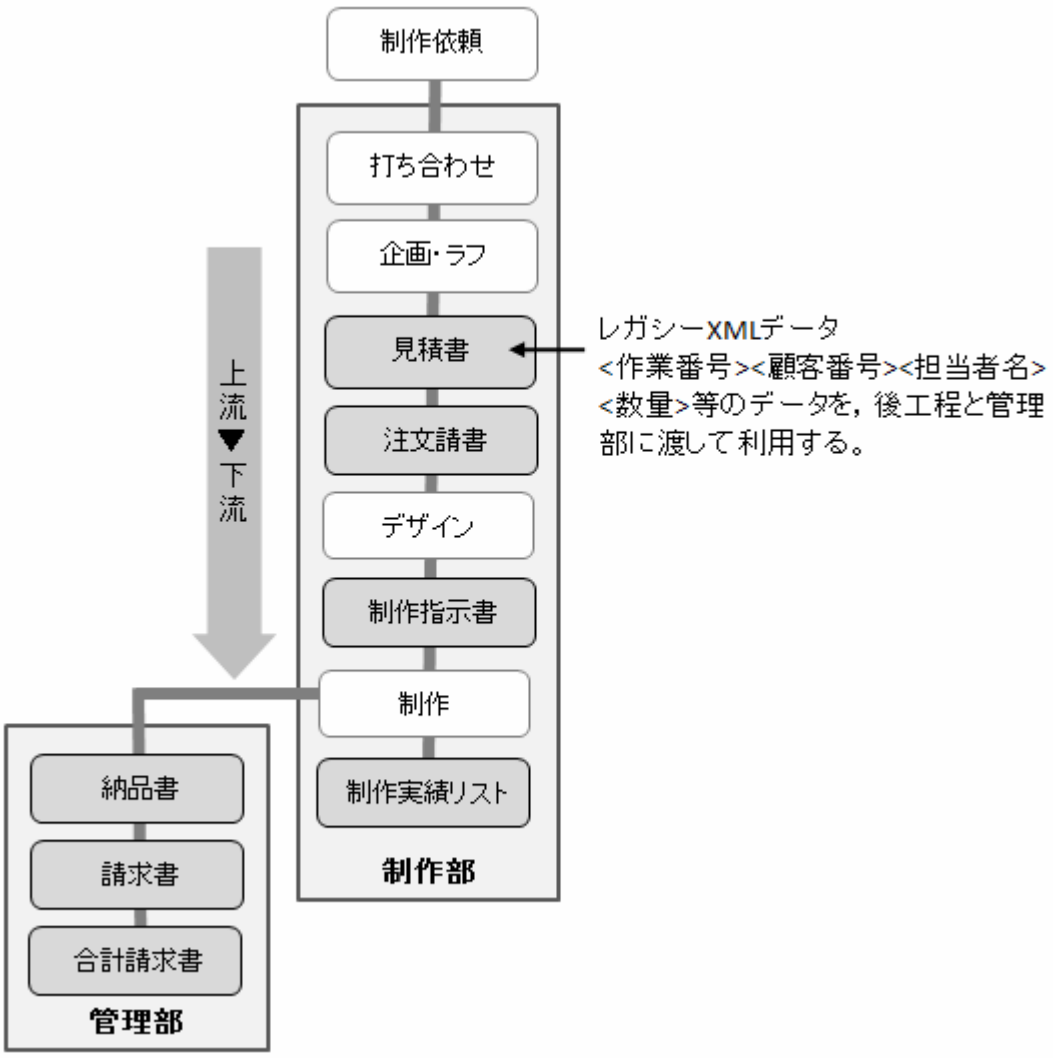
●図 4-2 SGML，HTML，XML，XHTML の関係



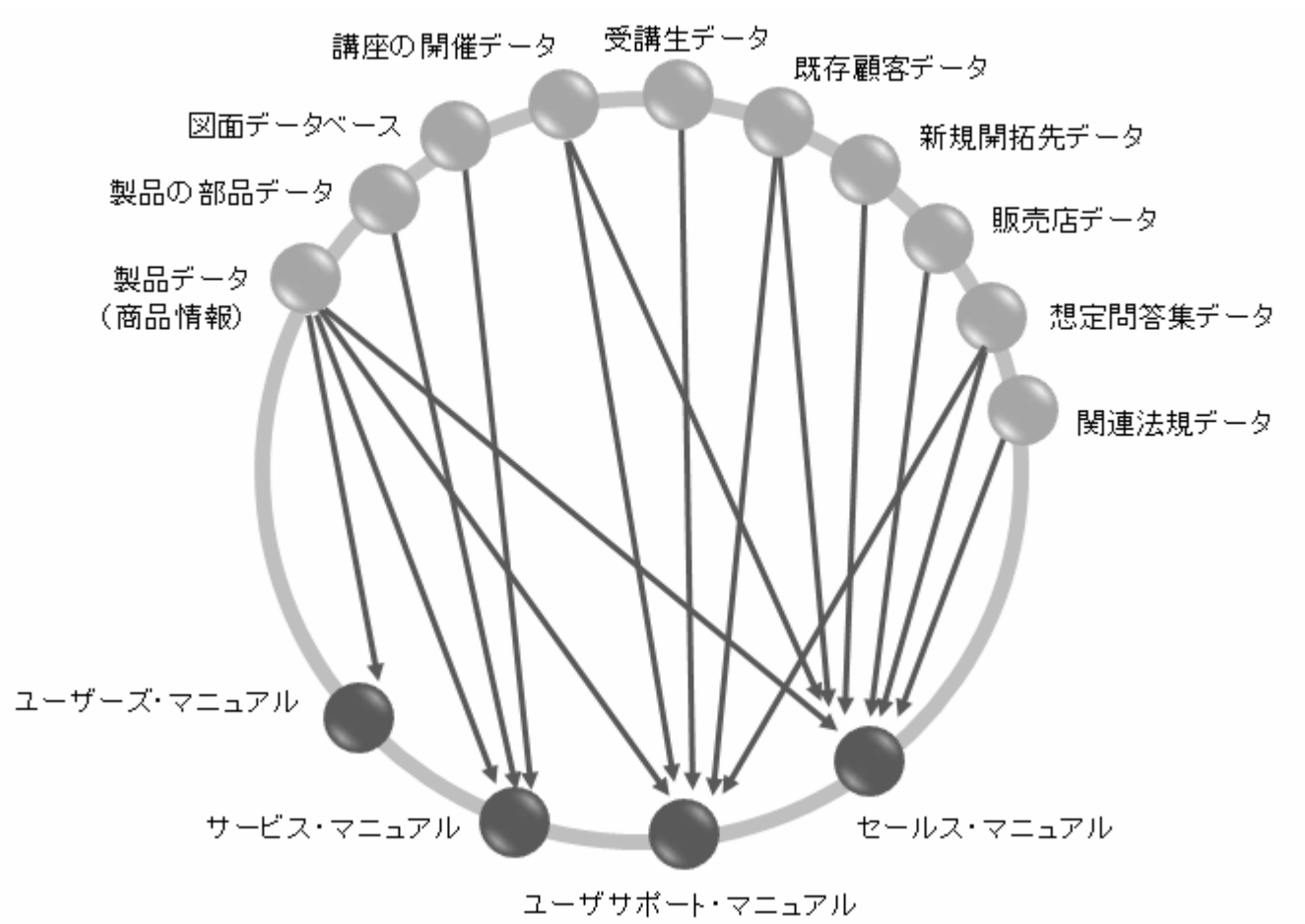
●図 4-3 XML は，CALS と SGML のメリットを受け継ぎ，利用時間と利用空間を拡大した。



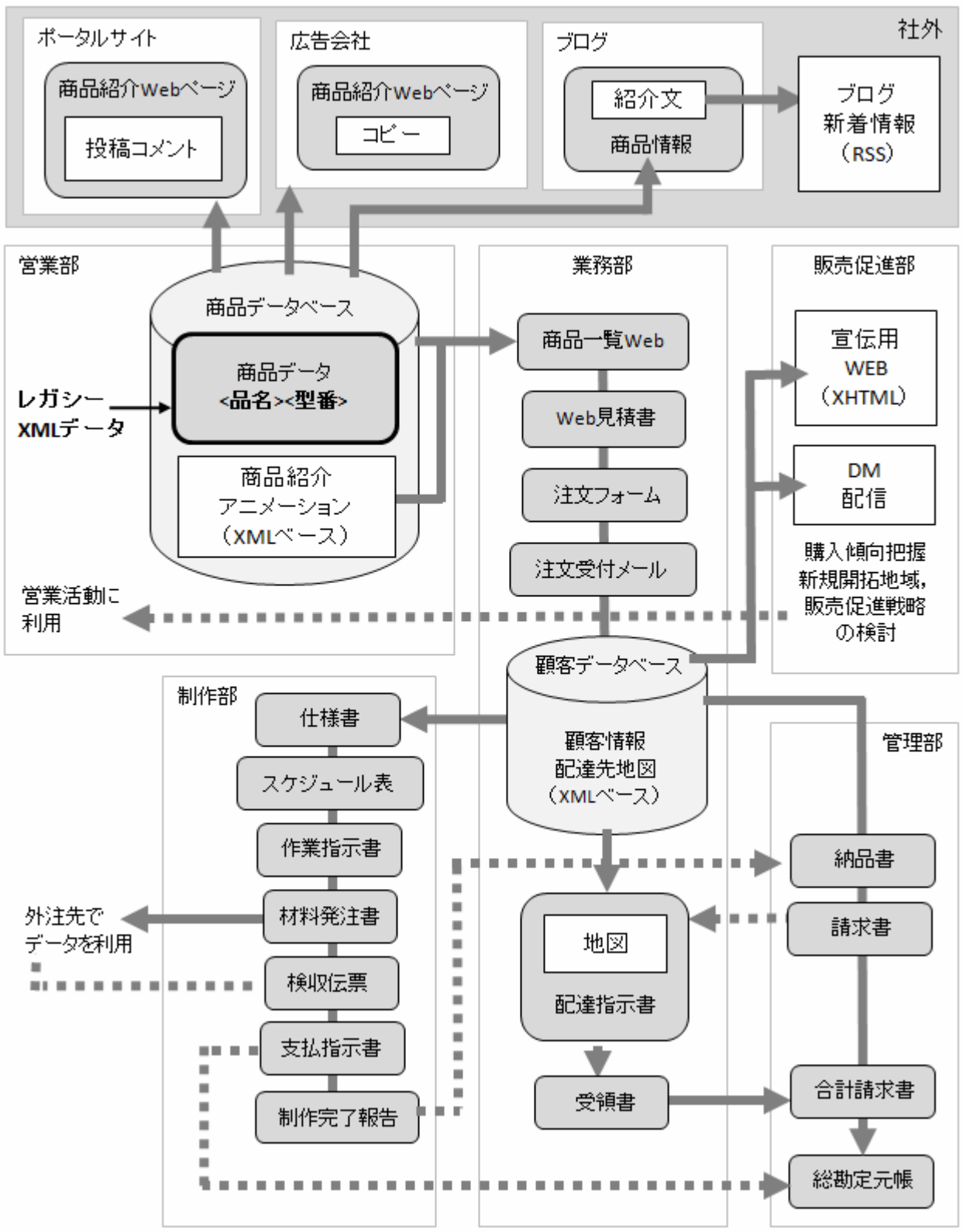
●図 4-4 見積書から合計請求書発行までデータを流用する最も簡単な例。



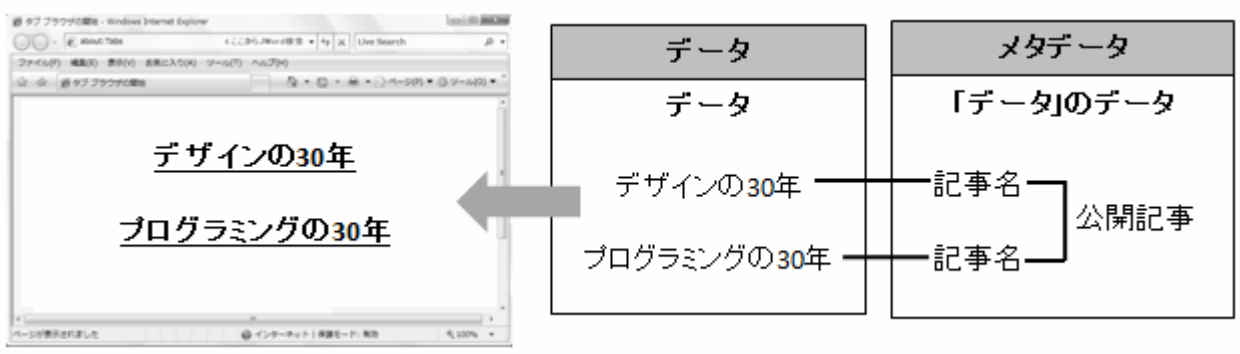
●図 4-5 複数の異なるレガシーXML データを，それぞれ 1 個の文書に利用する例。



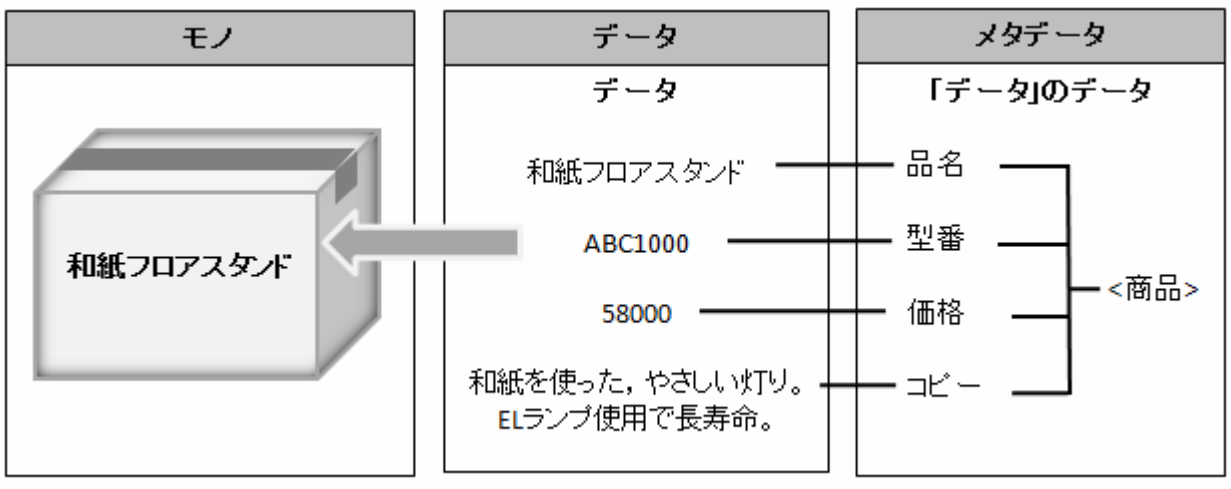
●図 4-6 レガシー商品データが利用範囲を拡大していく例。



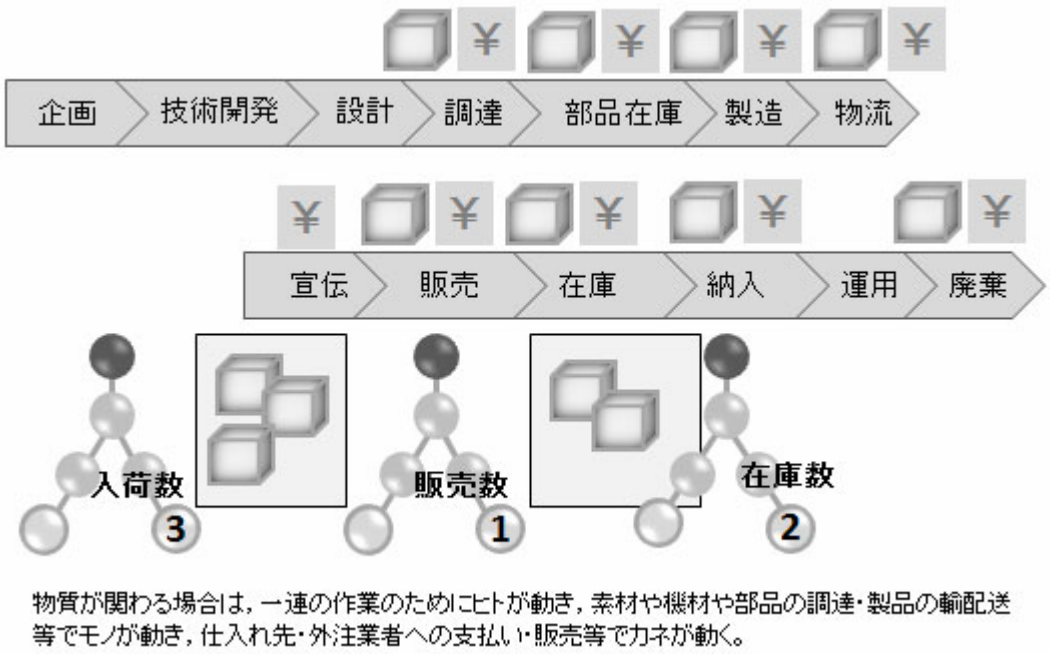
●図 4-7（第一章●図 1-19 再掲載）データと，データを表すメタデータ。



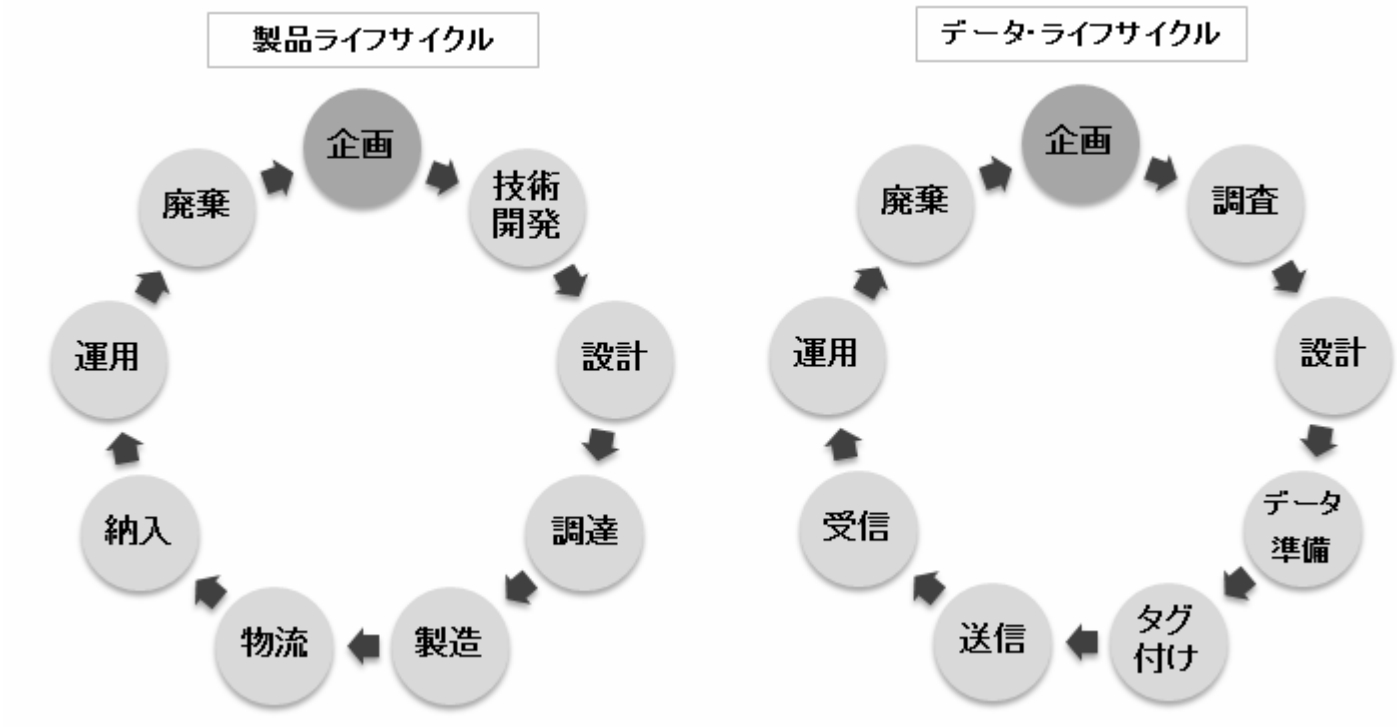
●図 4-8 物質を表すデータと，データを表すメタデータ。



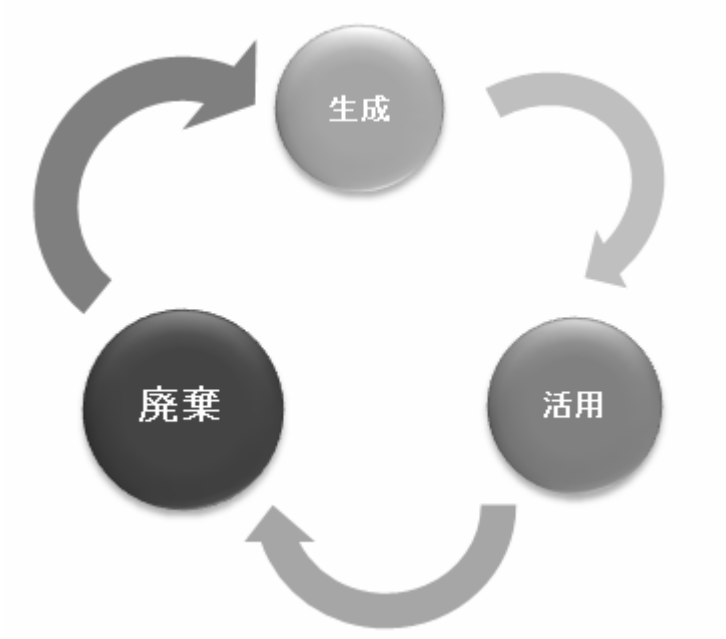
●図 4-9 XML データは，フィジカルな動きと連動して，逐次変化する。



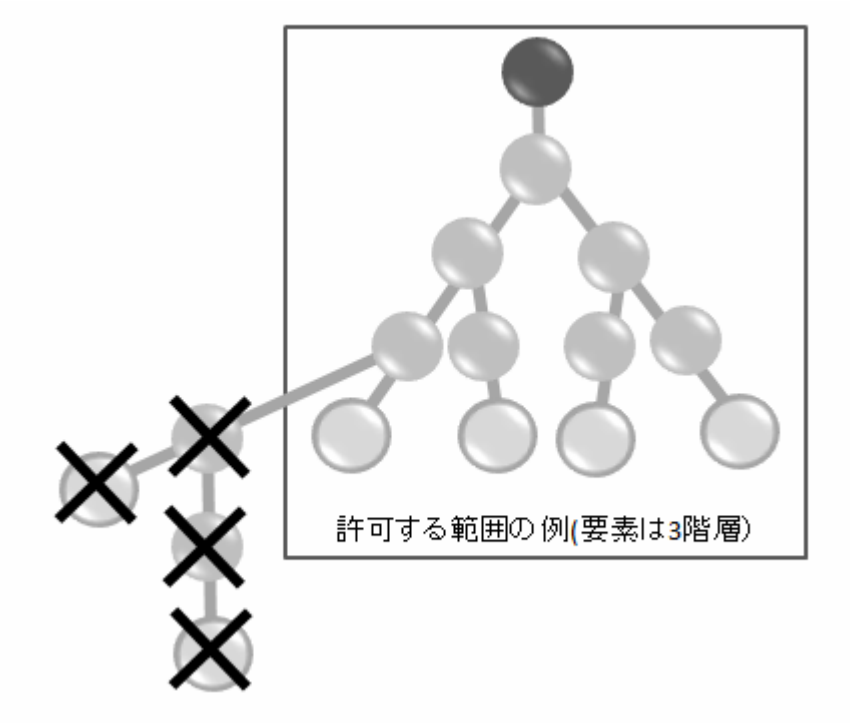
●図 4-10 製品ライフサイクル（生販物のロジスティクス）と，データ・ライフサイクル（信号のロジスティクス）



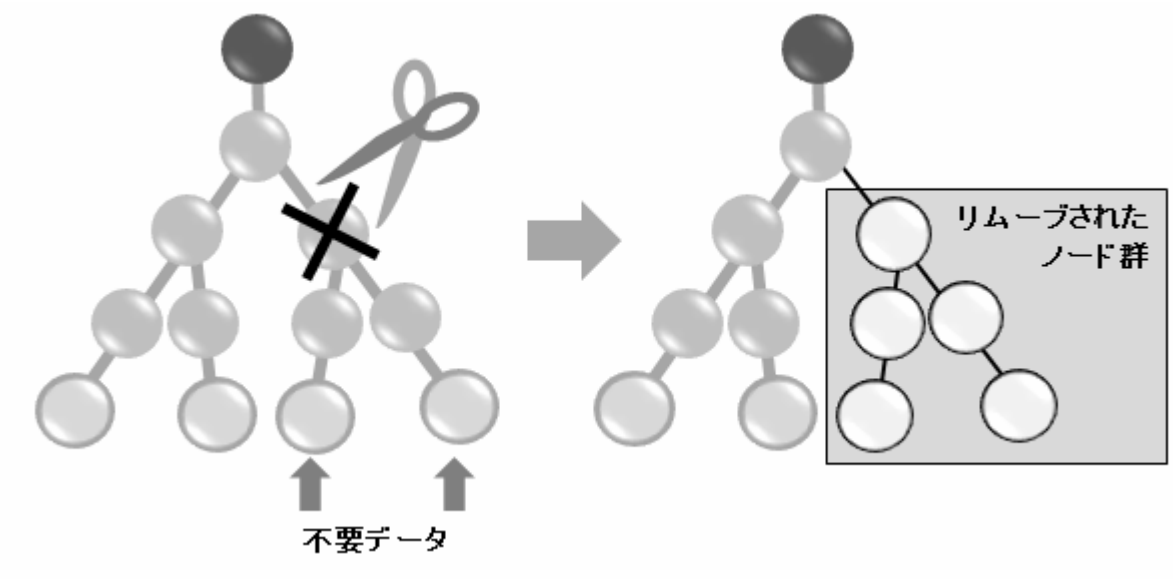
●図 4-11 廃棄までも重視した，生成・活用・廃棄の三位一体となった設計が必要である。



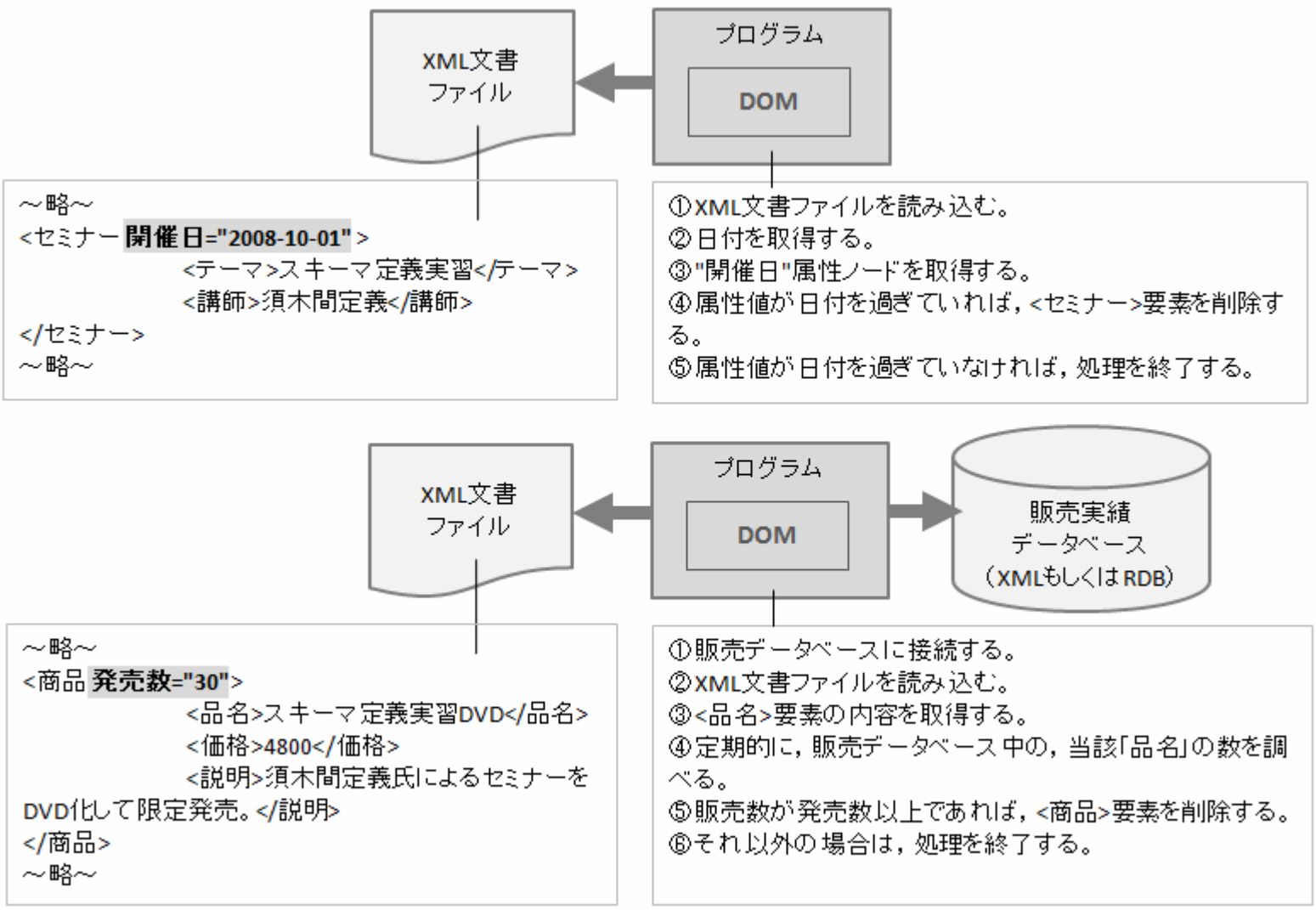
●図 4-12 木の枝葉を伸ばす範囲を決めて，ノード爆発を防ぐ。



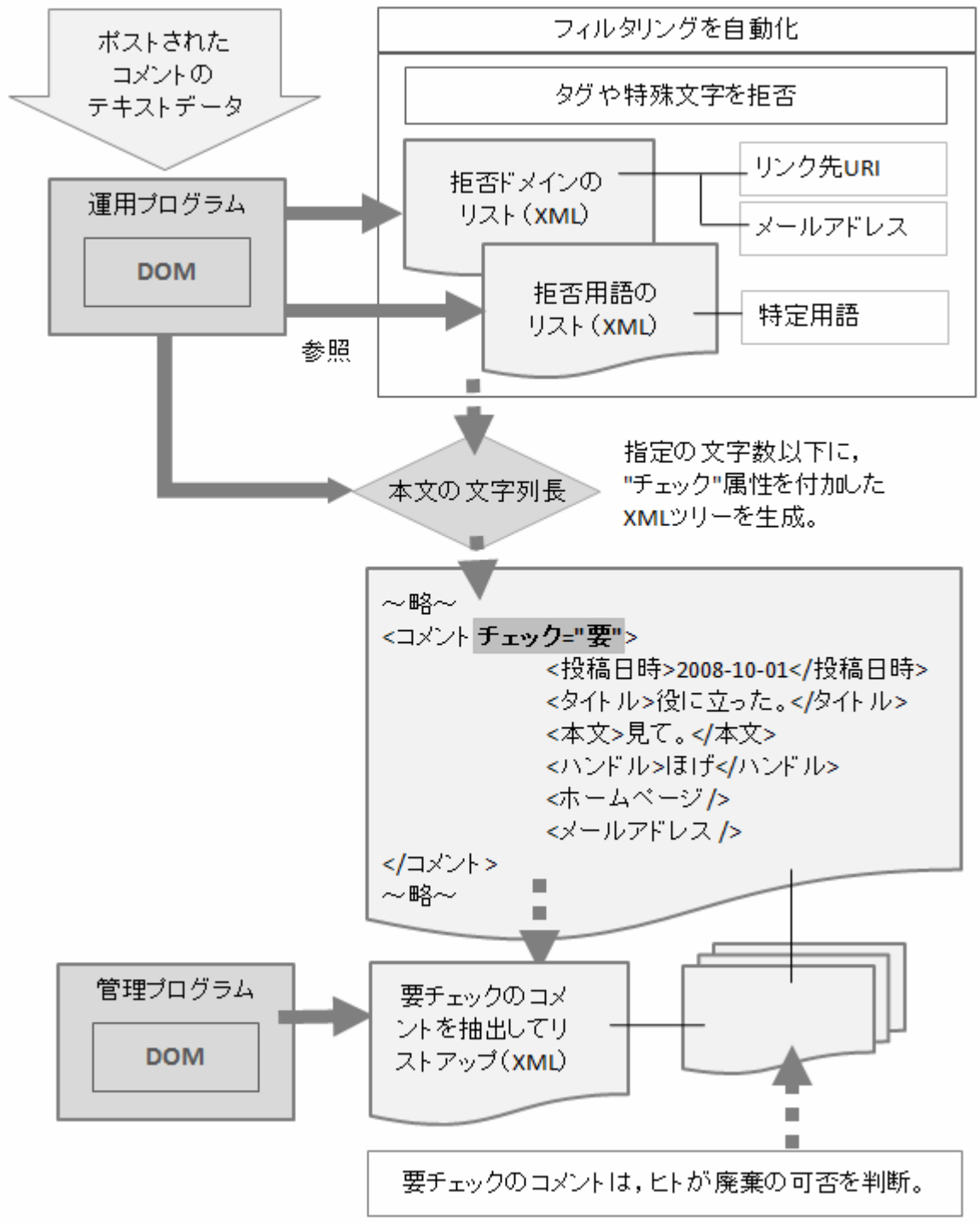
●図 4-13 迂回路がない木構造の特長を生かして，最小の処理で経路を断てるようにする。



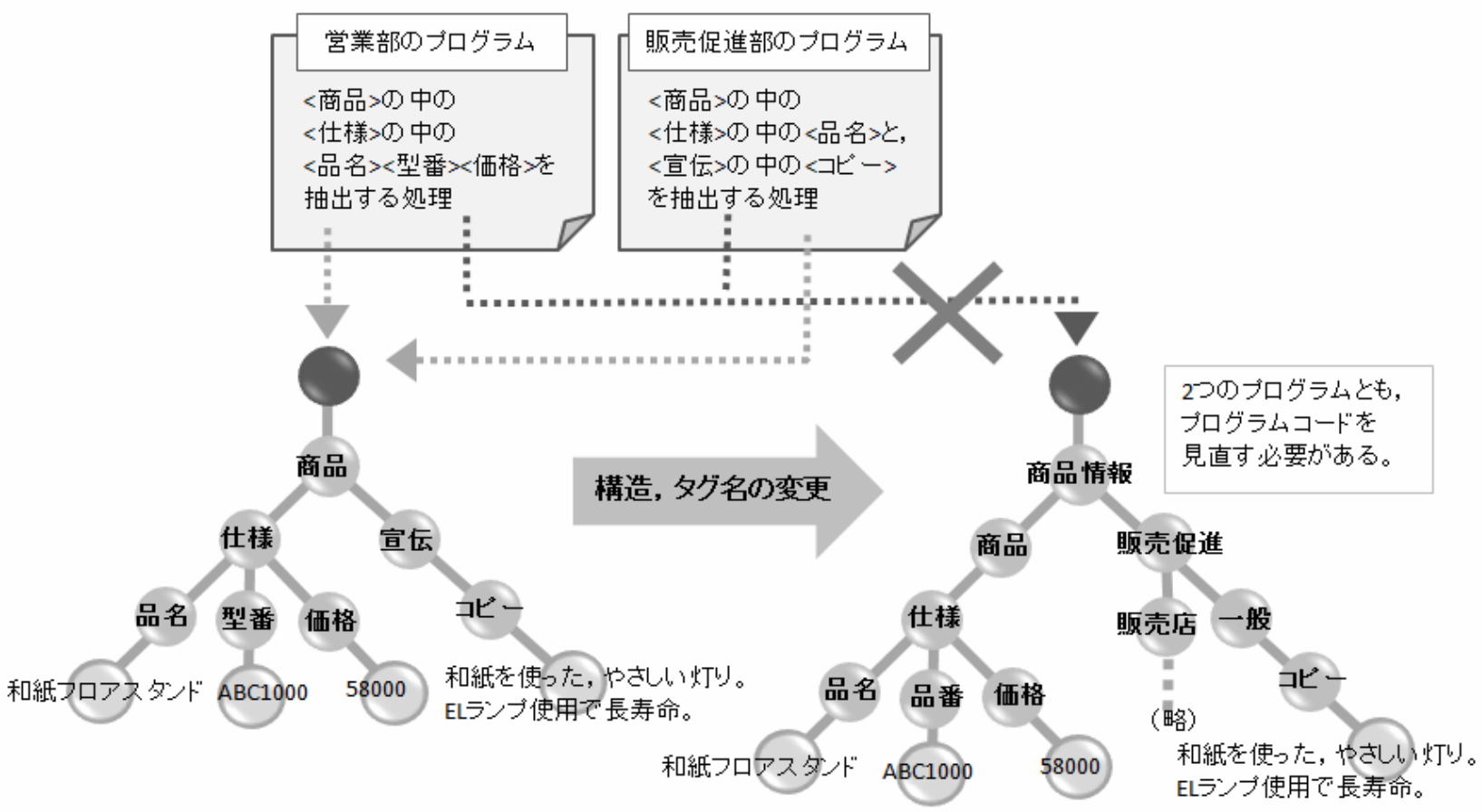
●図 4-14 廃棄の可否を判断できる属性を追加しておくと，自動的に処理できる。



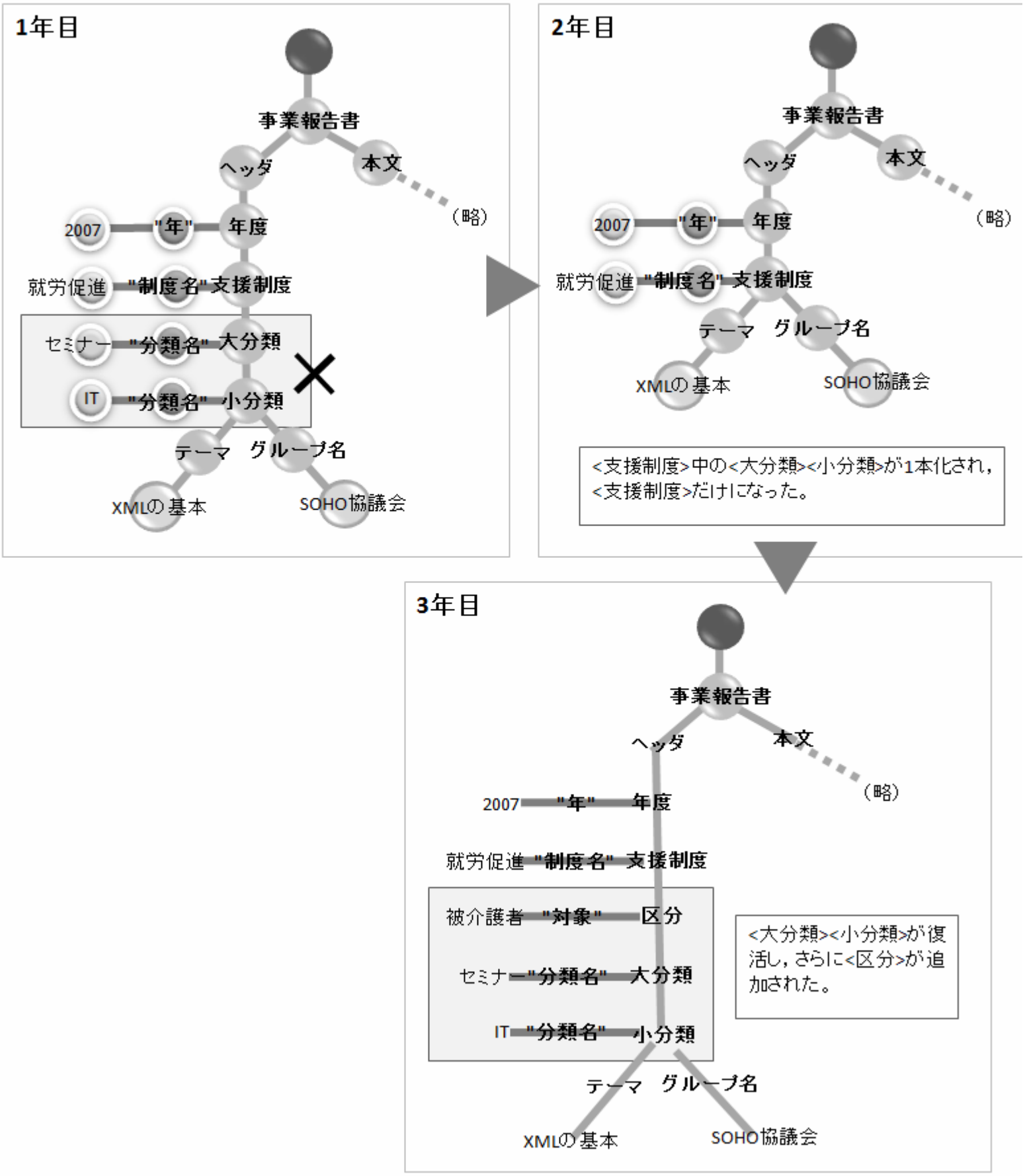
●図 4-15 コンピュータにより計算できない部分を，ヒトの判断で補える設計にする。



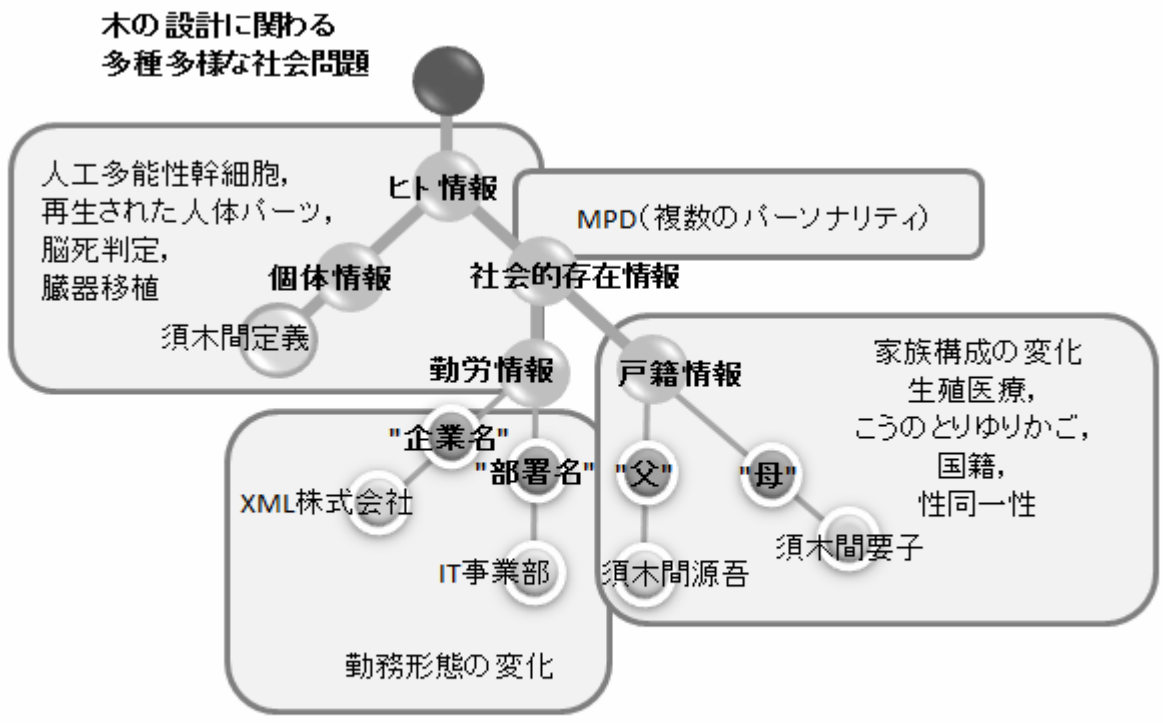
●図 4-16 利用中の木構造の中途階層の変更は，プログラムに影響を及ぼすことがある。



●図 4-17 社会情勢が設計に影響を及ぼす例



●図 4-18 一人のヒトの最小単位が変われば，木の構造も変わらざるをえない。



※この木はあくまで例であって，実務では，"企業名"ではなく，変更が生じない企業コードを用いるべきである。なぜなら，企業や商店の屋号は，一地域内では重複しないが，他の地域では重複する可能性があるからである。

第 5 章の図表

●図 5-1 ネイティブ XML 文書の基本的な設計工程

●表 5-1 ●図 5-1 中の広義の設計における作業

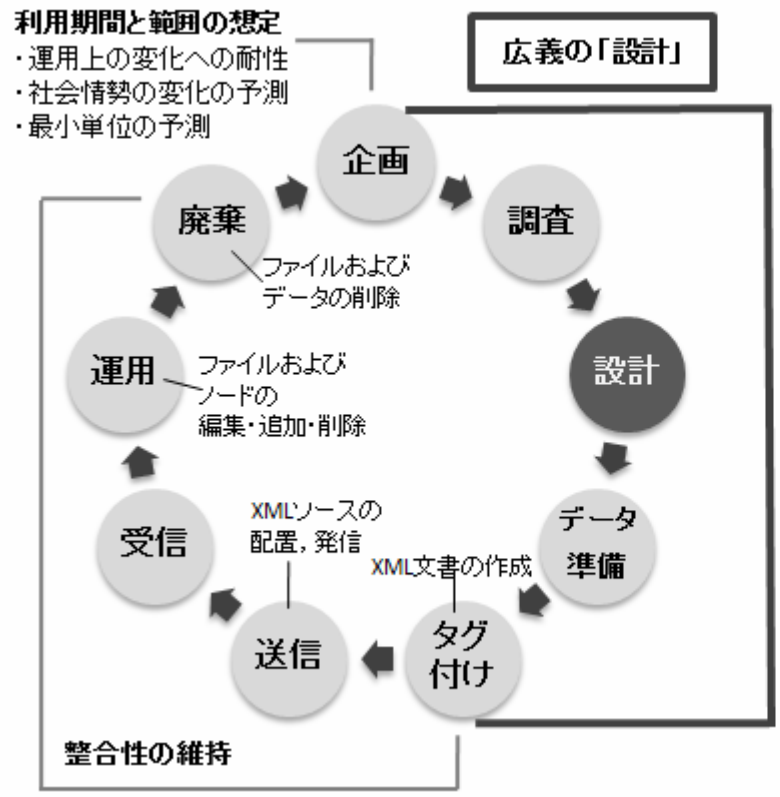


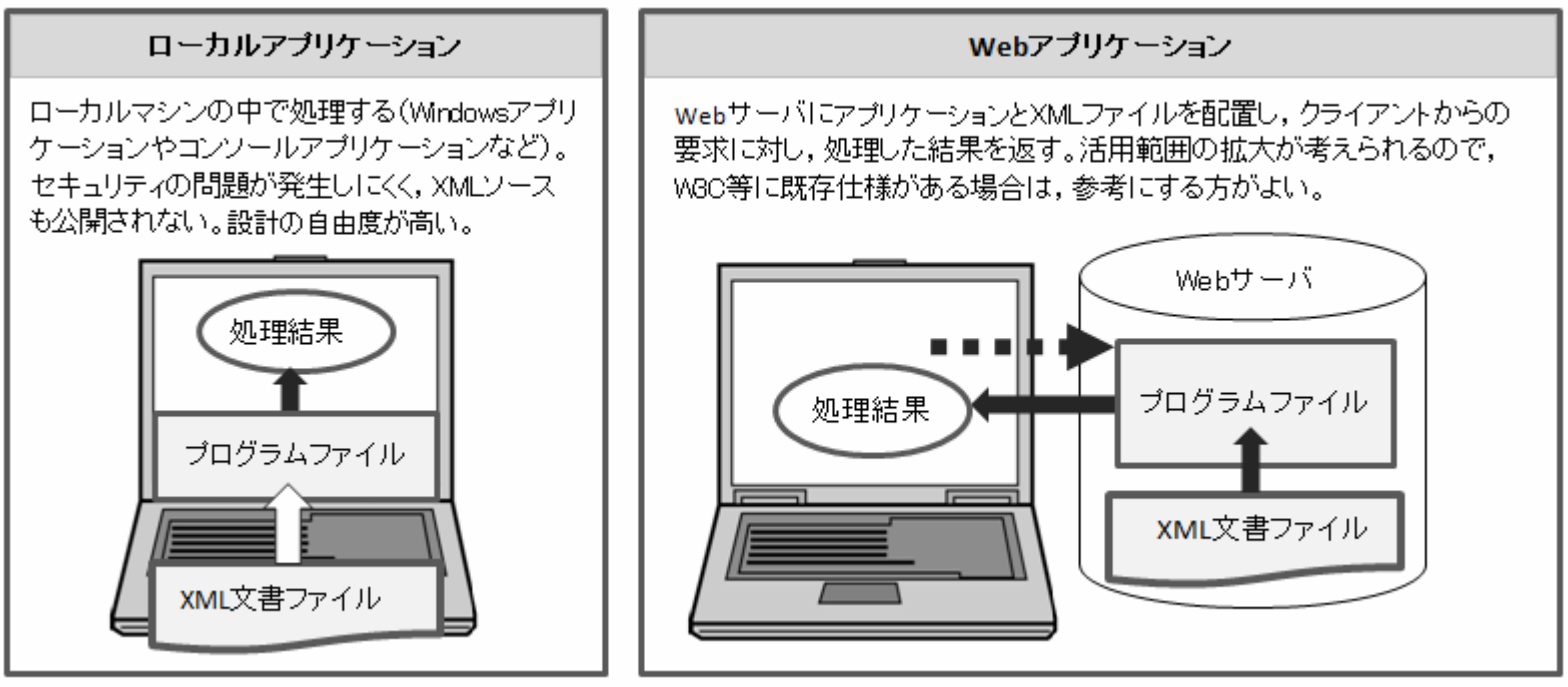
表5−1 ★図5−1 中の広義の設計における作業

企画	企画書作成
	処理アプリケーションの動作要件
	ミドルウエア導入の判断
	作成・処理のための仕様の決定
調査	設計用データの準備
	既存スキーマの調査
	設計のルール作り
	XML文書の作成方法決定
設計	木構造の設計
	文書内における整合性の確認
	文書ファイル群の構成の検討
	ユーザインタフェースの設定と適用
	複数ファイル間における整合性の確認
	タグ名の確定
	仕様書の作成
データ準備	実データの準備
タグ付け	XML文書ファイル生成用アプリケーションの開発
	XML文書ファイルの作成

●表 5-2 ネイティブ XML 文書設計の「企画」工程

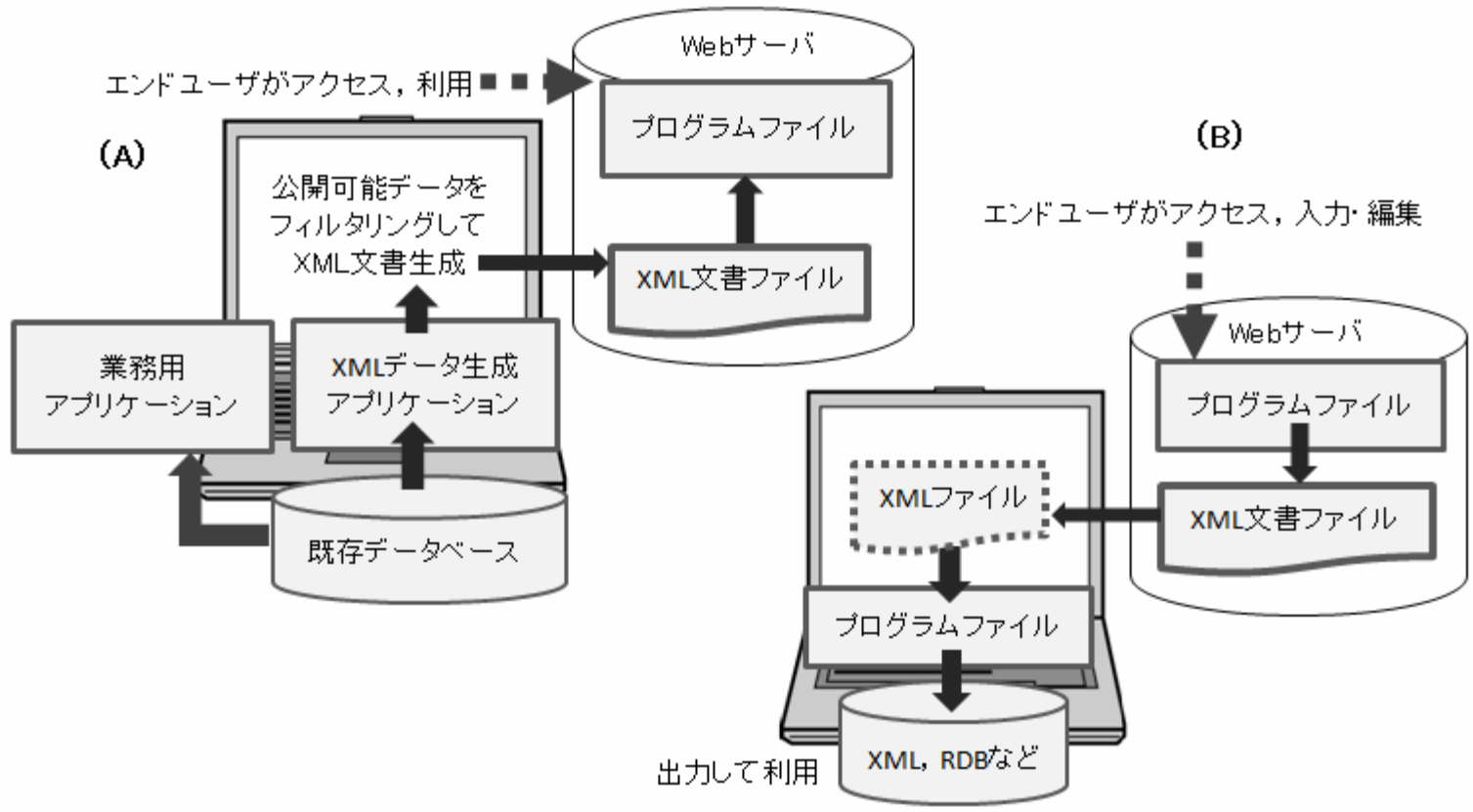
企画	
工程	検討・決定事項
企画書作成	利用対象（ユーザ層，スキル）
	運用範囲（時間，空間）
	XML化の目的（RDBと比較しての優位性，XML化による利点と問題点）
	期待効果（活用方法，安定性，拡張性，達成目標）
処理アプリケーションの動作要件	使用するXML仕様のバージョン
	アプリケーションの種類（ローカルアプリケーション，Webアプリケーション，Webサービス，モバイルアプリケーション，AIRやSilverlightなどによるRIA，ガジェット等）
	ハードウエアの動作要件（CPU，メモリ，OS，フレームワーク，それぞれのバージョン）
	ユーザ側の環境（パソコン，携帯端末，PDA等。OS，画面解像度，回線，ブラウザとそのバージョン，プラグインとそのバージョン）
	XML文書ファイルを蓄積する環境（サーバOS，フレームワーク）
ミドルウエア導入の判断	セキュリティ，耐障害性，排他制御，インデックス作成，処理対象文書のサイズなどから検討
作成・処理のための仕様の決定	XML仕様のバージョン
	XML周辺仕様とそのバージョン

●図 5-2 XML データ処理アプリケーションの種類を検討する



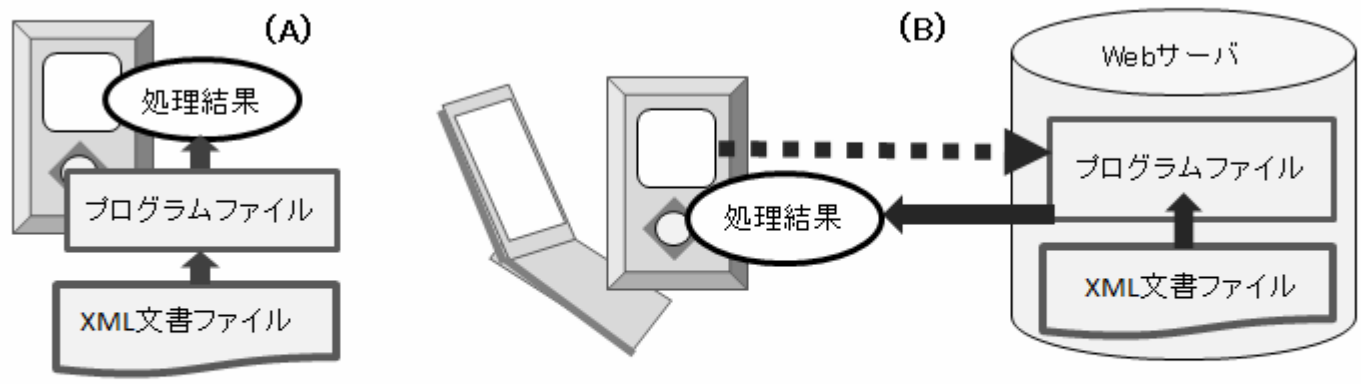
ローカルアプリケーションと連携するWebアプリケーション

(A) 既存ローカルアプリケーションで利用中の(主に)RDBのデータから，公開可能なデータを抽出してXML文書ファイルを作成する。タグ名は既存のフィールド名に準じた方がよい。
(B) Webアプリケーションから入力されたデータをもとにXML文書ファイルを作成し，ローカルアプリケーションで利用する場合は，最低限でも開発フレームワークの検証コントロールなどを利用したデータの検証が必要である。設計時には，データについても定義しておく(後述，★表5-6参照)。



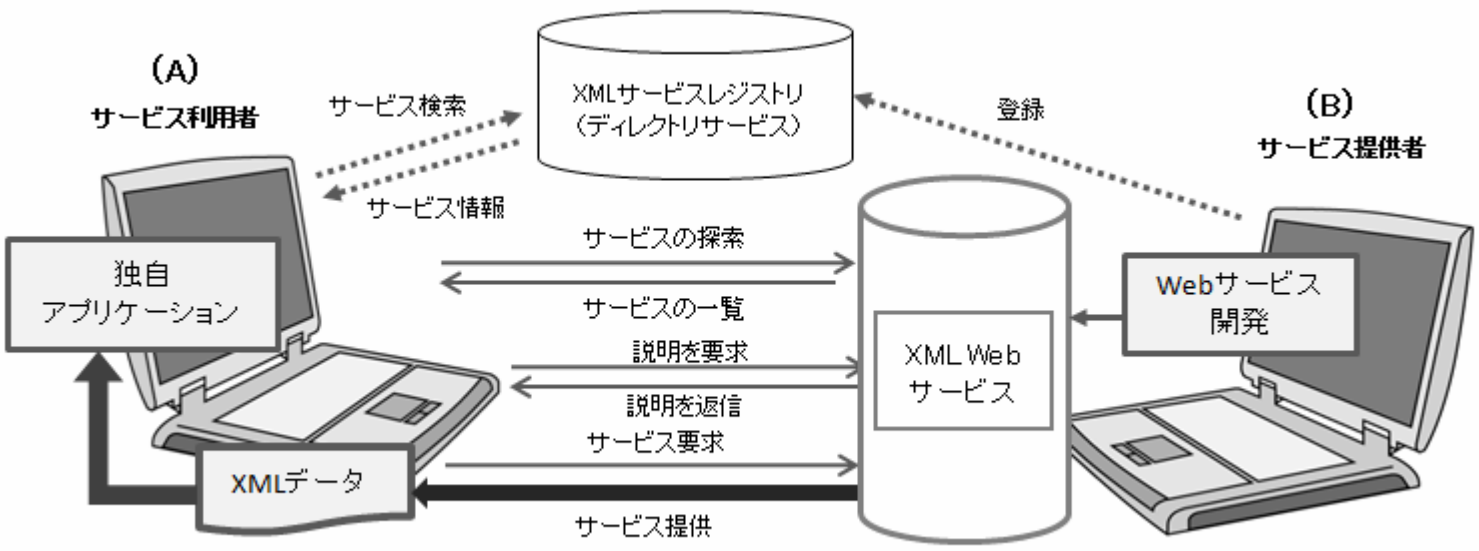
モバイルアプリケーション(PDA，携帯電話)

(A) 携帯端末内に配置したアプリケーションでXMLデータを
処理する場合は，携帯端末側のメモリの制約から，処理対象ノード数と階層の目安を設けて，その範囲内で設計する。
(B) ケータイ用Webアプリケーションで処理するXML文書の設計は，パソコン用Webアプリケーションの設計に準じる。表示スタイルの制約はXSLTスタイルシートの切り替えで対応すればよい。対応機種によっては文字コードの問題に注意する必要がある。

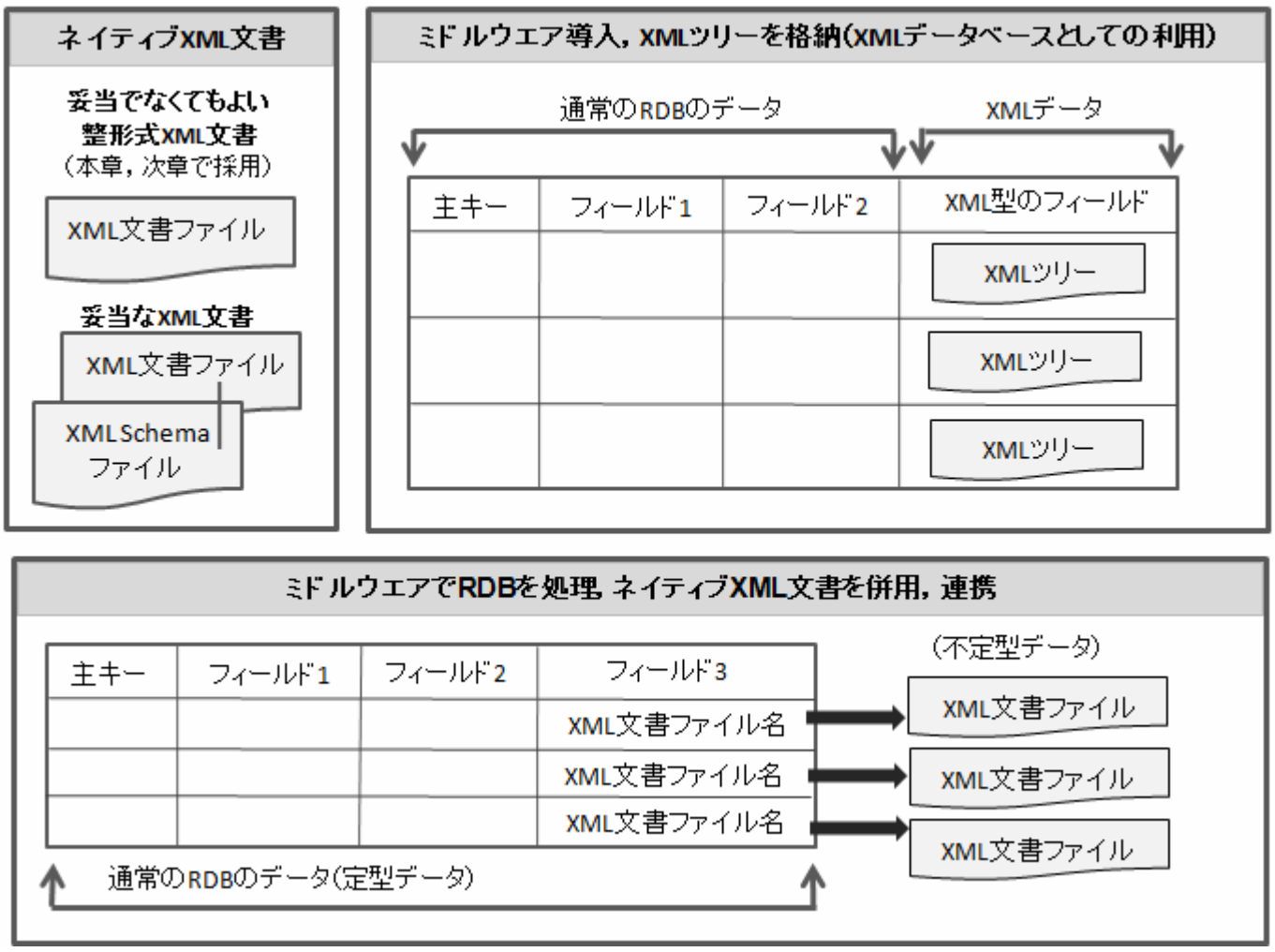


Webサービス

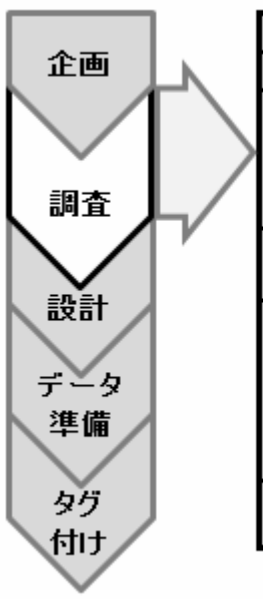
(A) Webサービスで返されたデータを利用する場合や，複数のXML Webサービスを利用したアプリケーションを開発する場合は，返されたXML文書の設計を踏襲するかカスタマイズして利用する。
(B) Webサービスの提供者側となる場合は，Webアプリケーションの設計に準じる。タグ名は英字の方がよい。



●図 5-3 ミドルウェアの導入について検討する



●表 5-3 ネイティブ XML 文書設計の「仕様調査」工程

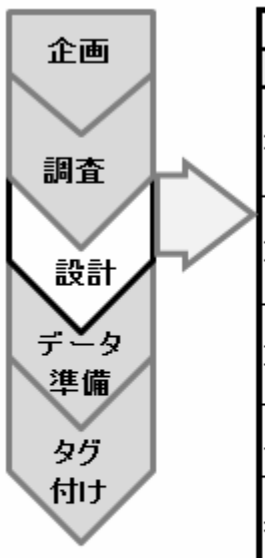


調査	
工程	作業
設計用データの準備	サンプル・データ準備，なければダミーデータを作成
	データの内容，および更新方法，最終的な予想ノード数，データの機密性の確認
	データの再利用の可能性，利用頻度の確認
既存スキーマの調査	W3C仕様の調査，各業界団体の仕様の調査
	既存スキーマの利用，カスタマイズ，新規定義の決定
設計のルール作り	検証の必要性の有無，検証方法を決定
	名前空間URIと接頭辞の決定
	文字コードと名前付けルールを決定
	ルート要素名を決定
	暫定的なメタデータ名を仮定義
XML文書の作成方法決定	レガシーXMLデータの作成方法を決定

●表 5-4 名前付けルール

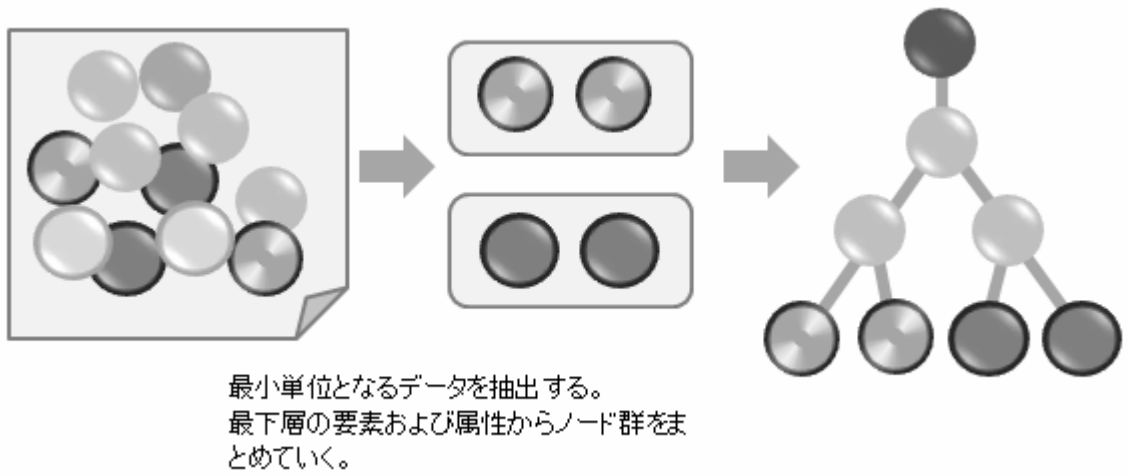
ルール	説明	例
Pascal形式	3文字以上の識別子について，1文字目と，それに続く各単語の最初の文字を大文字にする。 ただし，続く単語がない場合は，1文字目のみ大文字とする。例：<Items>	<OrderDate >
Camel形式	1文字目は小文字で，それに続く各単語の最初の文字を大文字にする。	<orderDate>
大文字	2文字以内のものについて，すべて大文字にする。	<NO>
XHTMLに合わせる	すべて小文字にする。	<title>

●表 5-5 設計工程の各作業

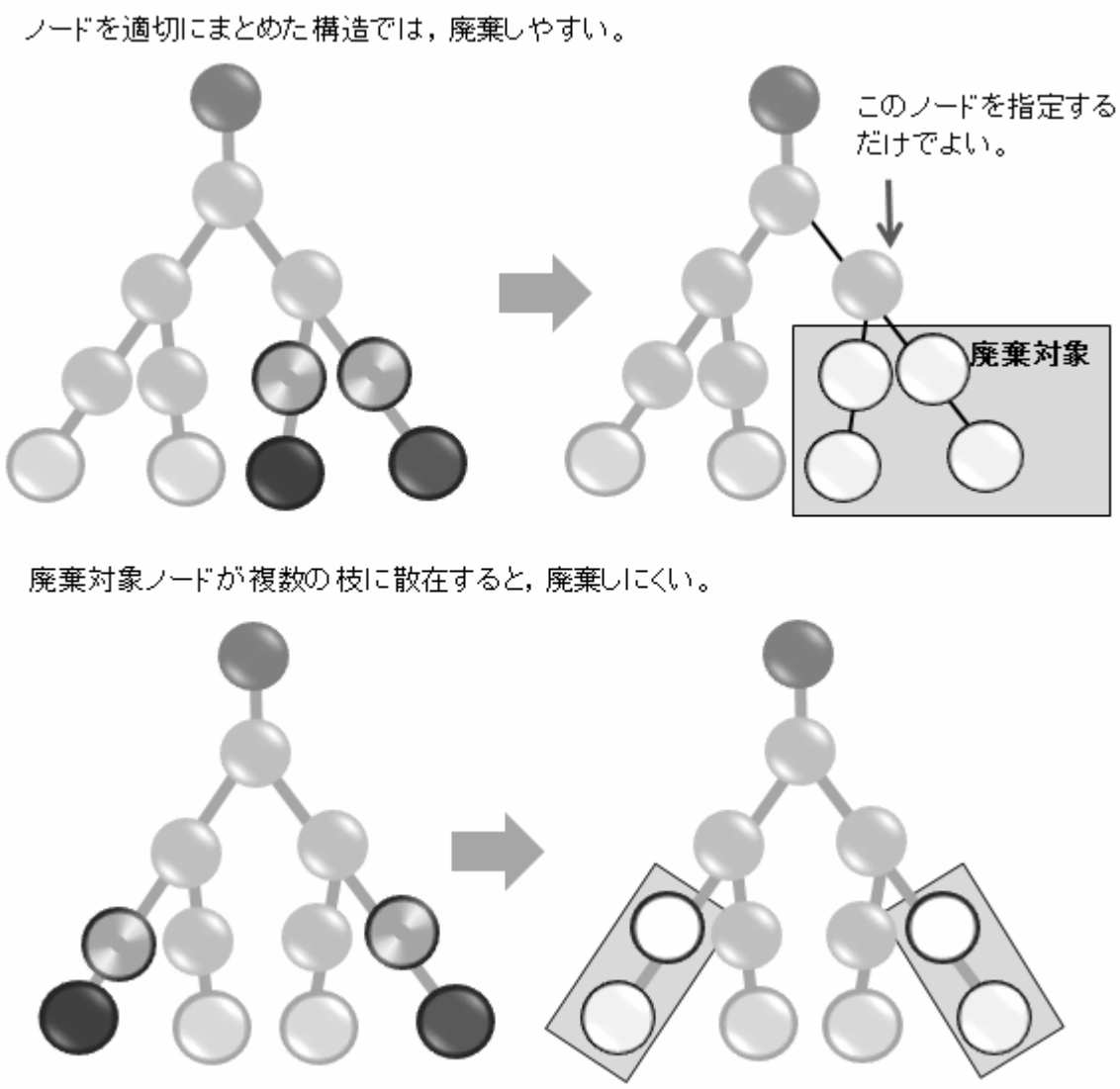


設計	
工程	作業内容
木構造の設計	ボトムアップ型で構造を検討
	ヒトによる理解とコンピュータによる計算のバランス
	技術以外の視点による構造の再確認
文書内における整合性の確認	ノードの編集・追加・削除と整合性の確認
	不整合を抑止する処理のルール作り
	ノード間の関係と処理実行のタイミングの確認
文書ファイル群の構成の検討	(複数ファイル構成を検討する場合)データファイルの構造
	(複数ファイル構成を検討する場合)ファイルの構成。管理ファイル使用の有無と，データファイルとの関係
ユーザインタフェースの設定と適用	設定ファイル使用の有無
	設定ファイルの用い方
複数ファイル間における整合性の確認	ファイルの変更，生成・追加，削除に伴う整合性の確認
	CRUDのタイミングの確認
	単一ファイルか複数文書ファイルか，ファイル構成を決定
タグ名の確定	タグ名の重複可能性の確認
	要素内容および属性値のルール作り
仕様書の作成	データ定義仕様書，サンプルXML文書ファイル，XML文書ファイルの作成方法，データ更新方法，設計までの経緯と決定理由

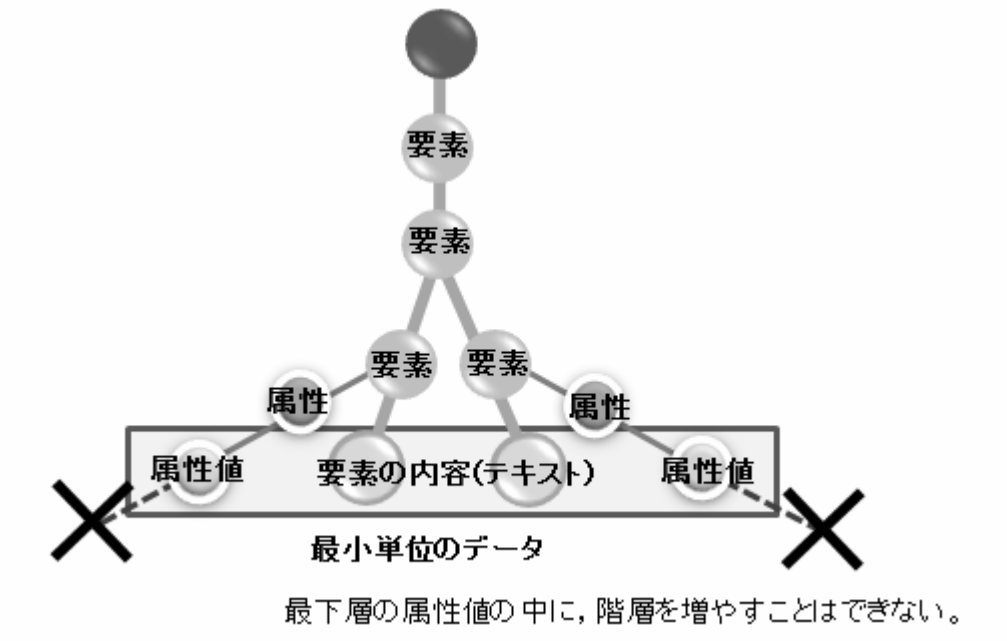
●図 5-4 ボトムアップで構造を考える



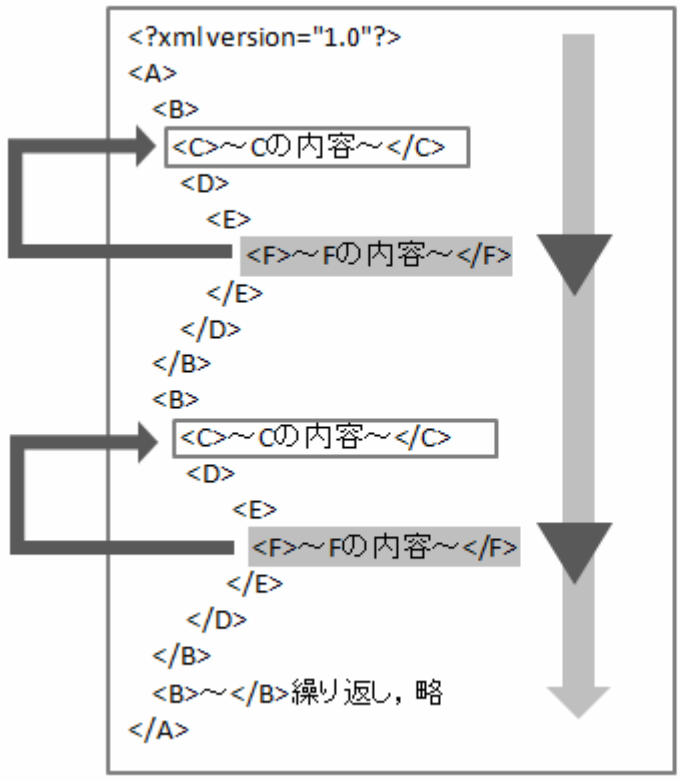
●図 5-5 廃棄しやすい設計は，ボトムアップ型で行う



●図 5-6 最小単位のデータは最下層の要素の内容か，従属する属性値に割り当てられる。



●図 5-7 走査効率の面から，「一歩下がって二歩進む」構造は避ける



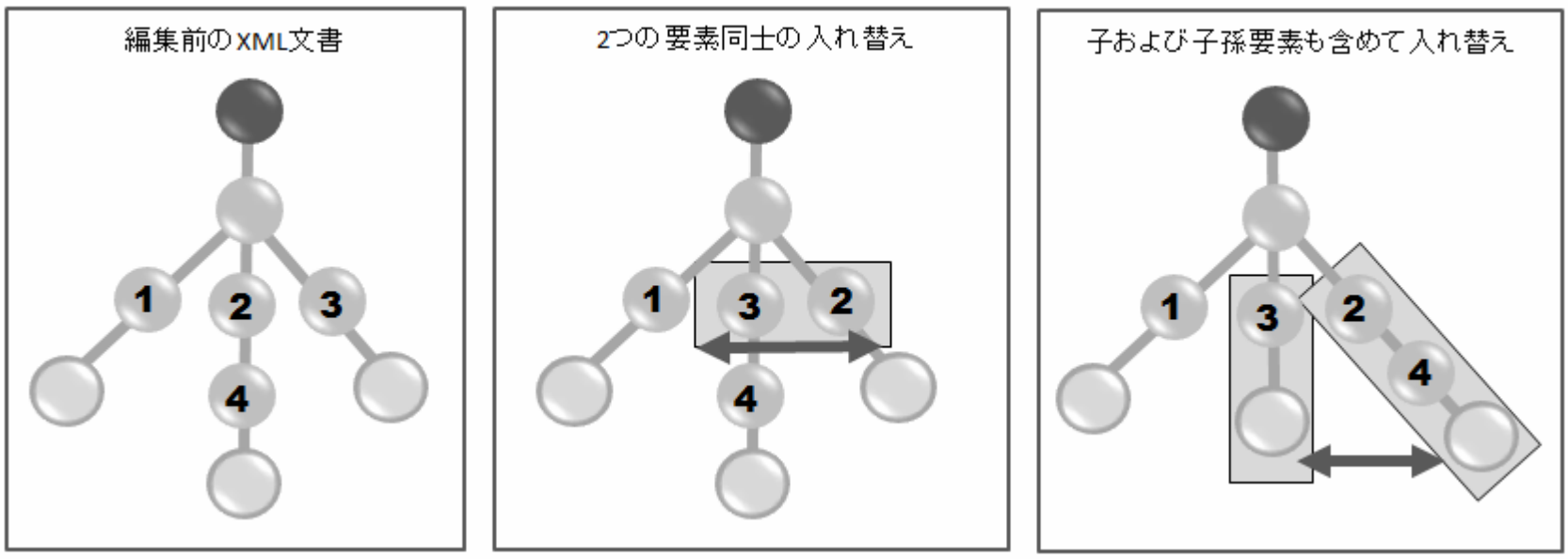
●表 5-6 大枠の設計ができた段階での確認事項

タグ名	暫定的なメタデータ名(この時点での要素名，属性名の最有力候補)
構造	出現順序
	繰り返し
	最低出現回数，最高出現回数
	空要素を許可するかどうか
データ	データ型
	値の下限，上限
	データ長
	デフォルト値
	選択肢の値
	処理のためのキーとするメタデータ

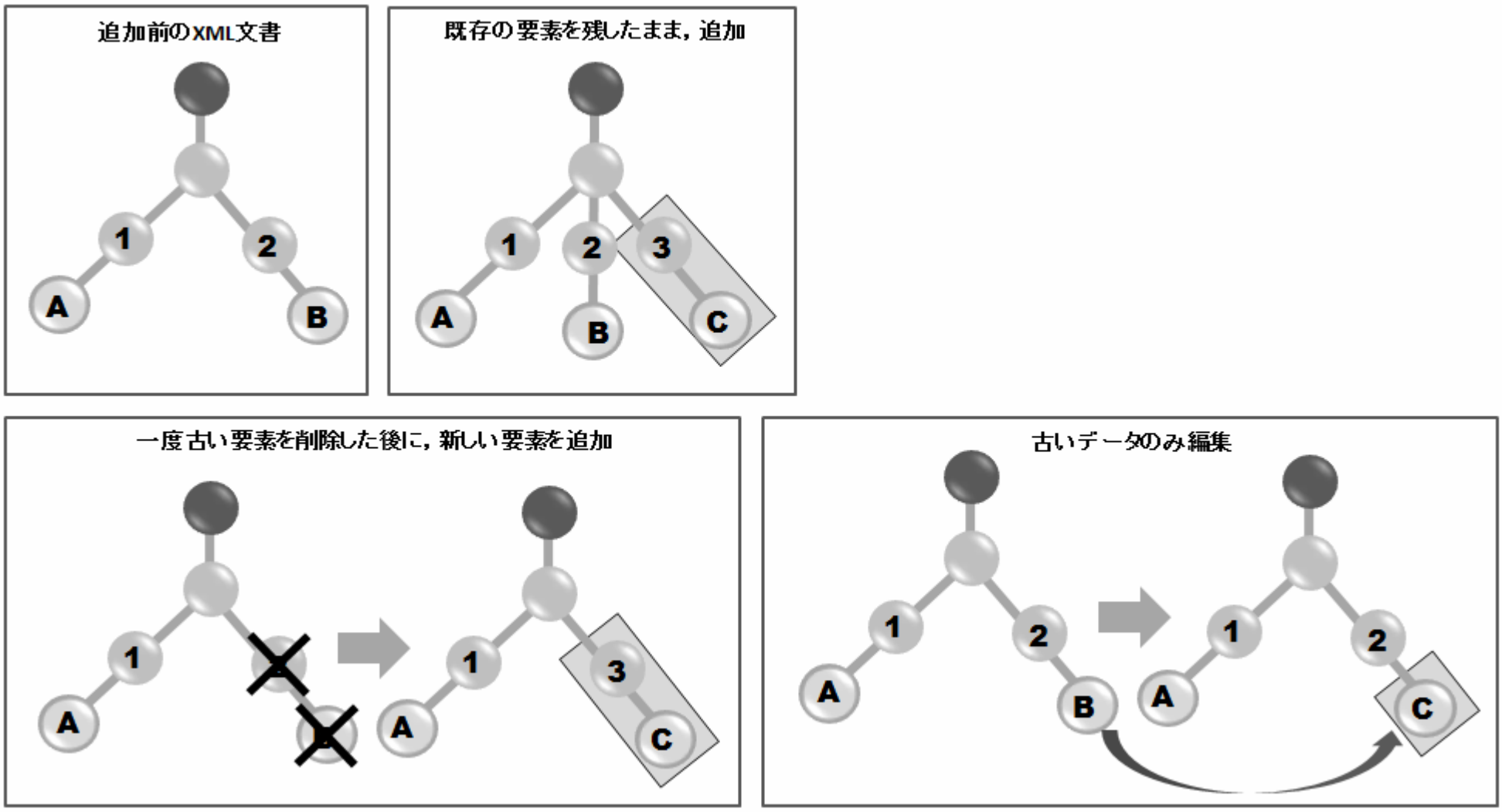
●表 5-7 要素・属性・テキストの編集・追加・削除

処理	操作対象	処理内容
編集	要素	順序を入れ替える。
		階層を入れ替える。
		要素名を変更する。
	テキスト	要素の内容テキストを変更する。
	属性	従属先の要素を変更する。
		属性名を変更する。
		属性値を変更する。
追加	要素	要素を末尾に追加する。
		要素を任意の階層の任意の位置に追加する。
	テキスト	空要素にテキストを追加する。
	属性	属性を任意の要素に追加する。
		値のない属性に値を追加する。
削除	要素	要素を削除する。要素の内容テキストや属性や子孫要素がある場合は，それらも削除する。
		要素を削除する。要素の内容テキストや属性がある場合は，それらも削除する。子孫要素は削除しない。
	テキスト	要素の内容のみ削除する。つまり空要素にする。
	属性	属性を削除する。
		属性値を削除する。属性はあるが値がない状態にする。

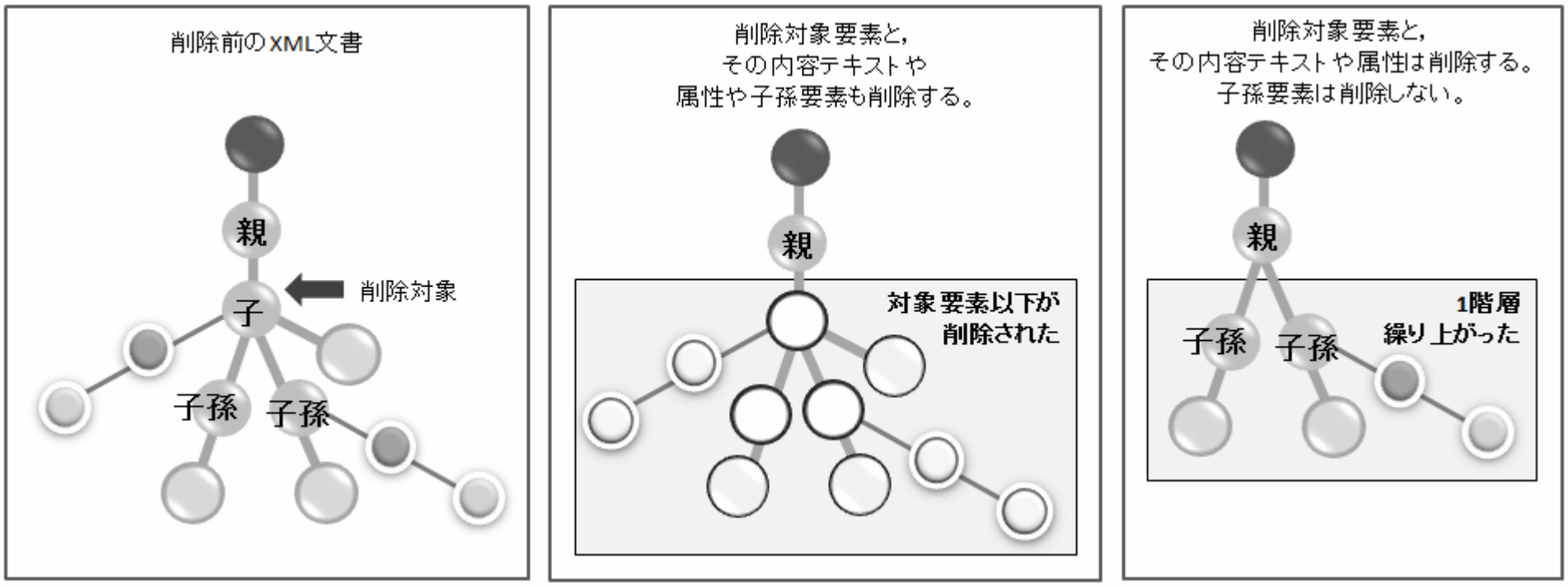
●図 5-8 要素の順序の入れ替え



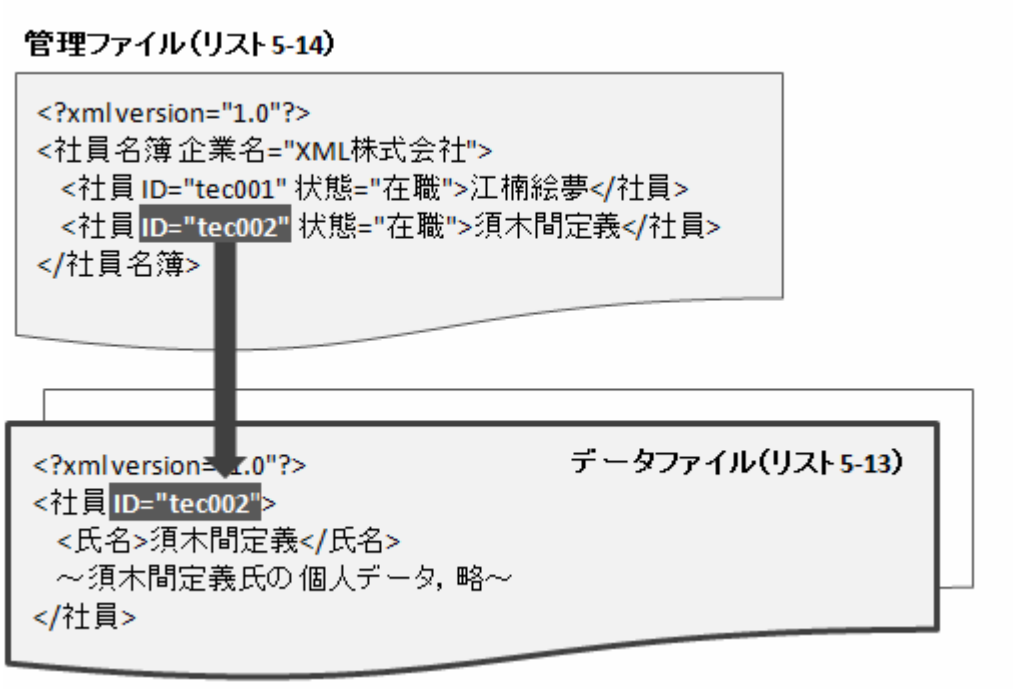
●図 5-9 追加にあたって既存の要素を残すかどうかを決めておく



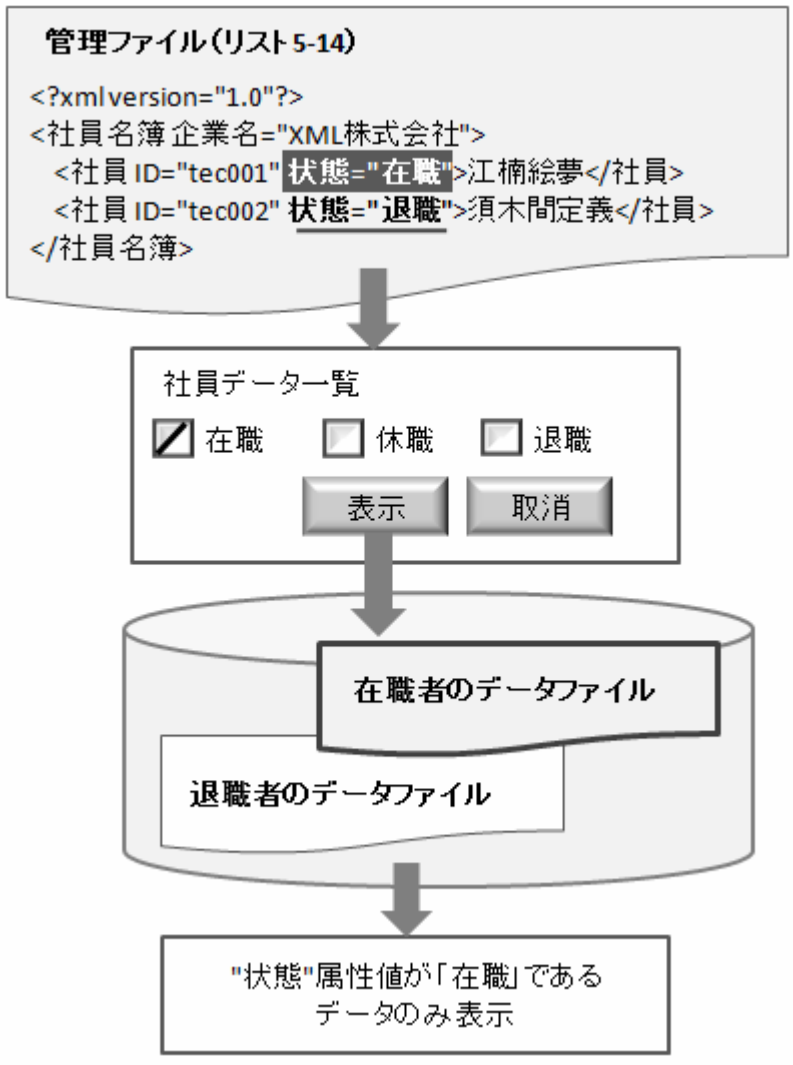
●図 5-10 削除対象要素の子孫を削除する場合と，削除しない場合



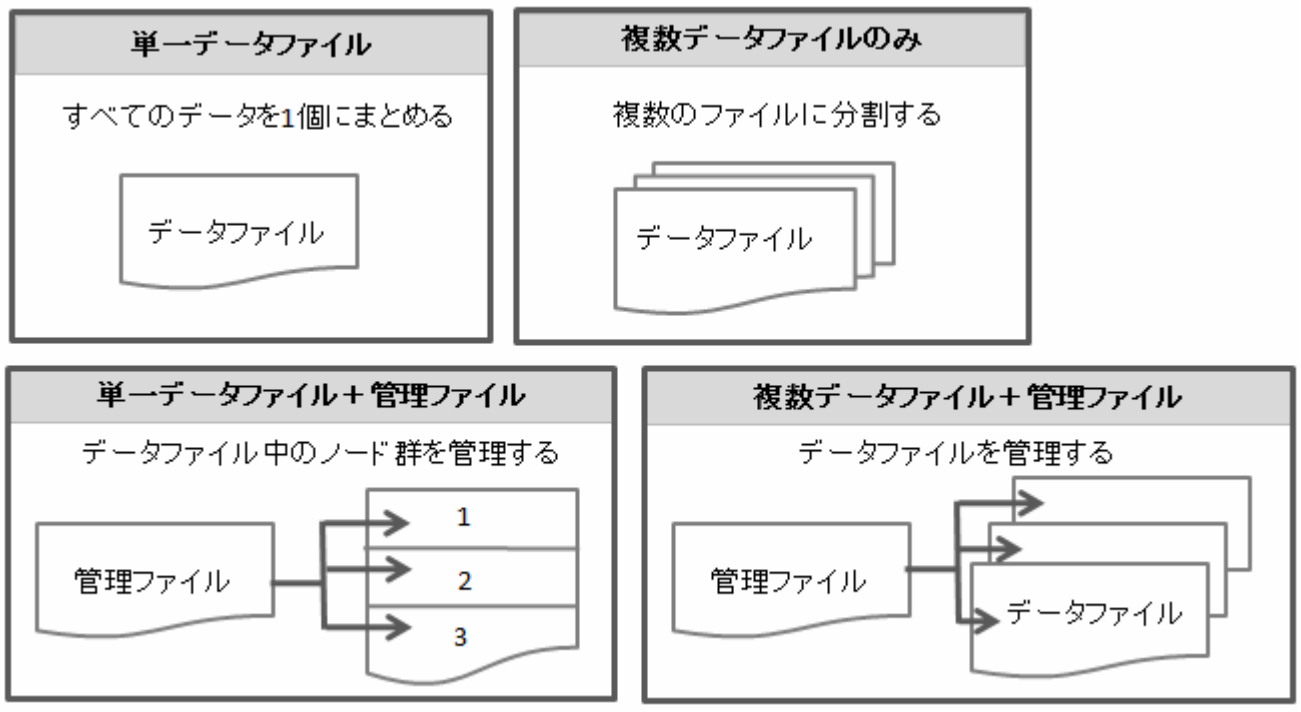
●図 5-11 ●リスト 5-13 のデータファイルと，●リスト 5-14 の管理ファイルを，"ID"属性値によって連携処理。



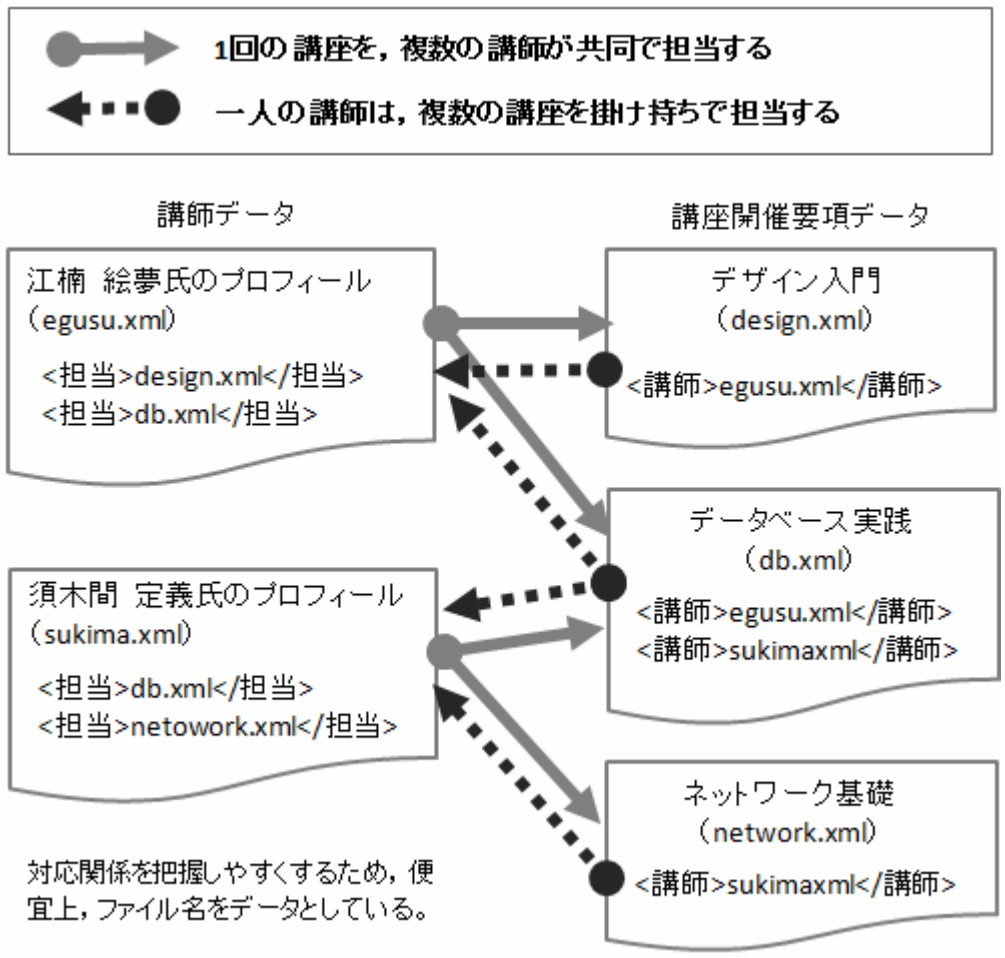
●図 5-12 退職後にデータファイルが蓄積されたままであっても，管理ファイルを用いてフィルタリングできる。



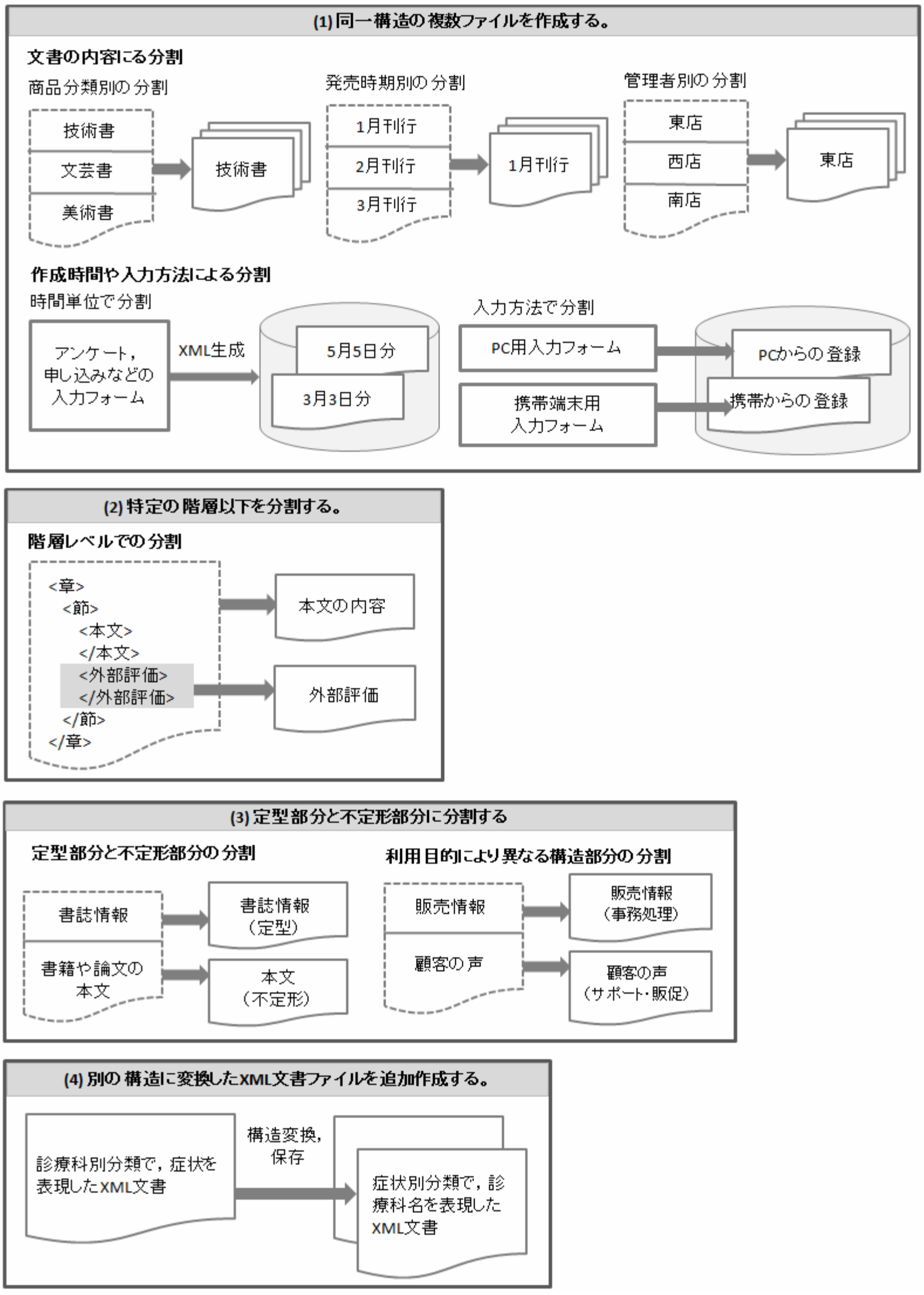
●図 5-13 XML 文書ファイル群の構成



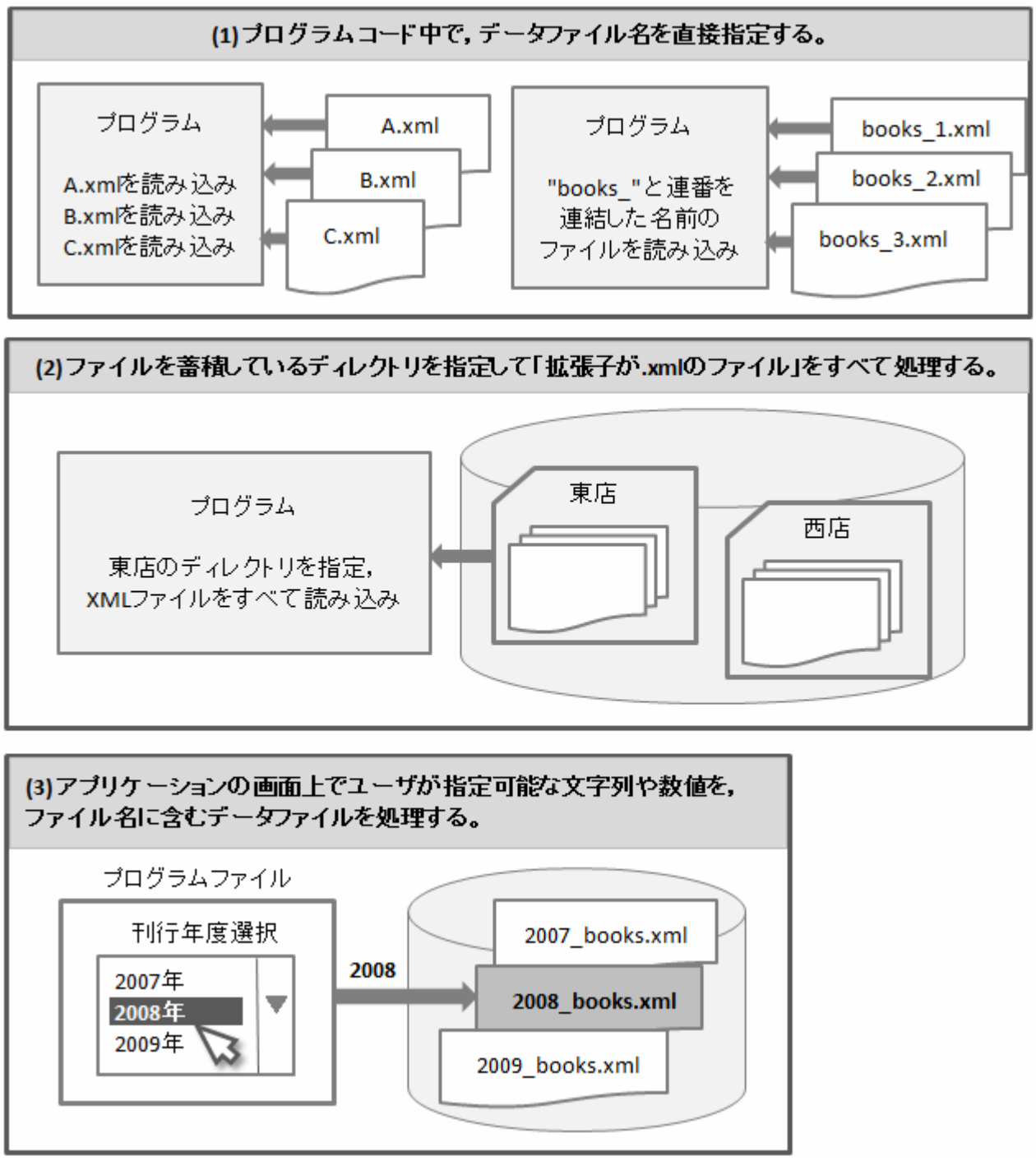
●図 5-14 複数ファイル間の多対多対応の関係



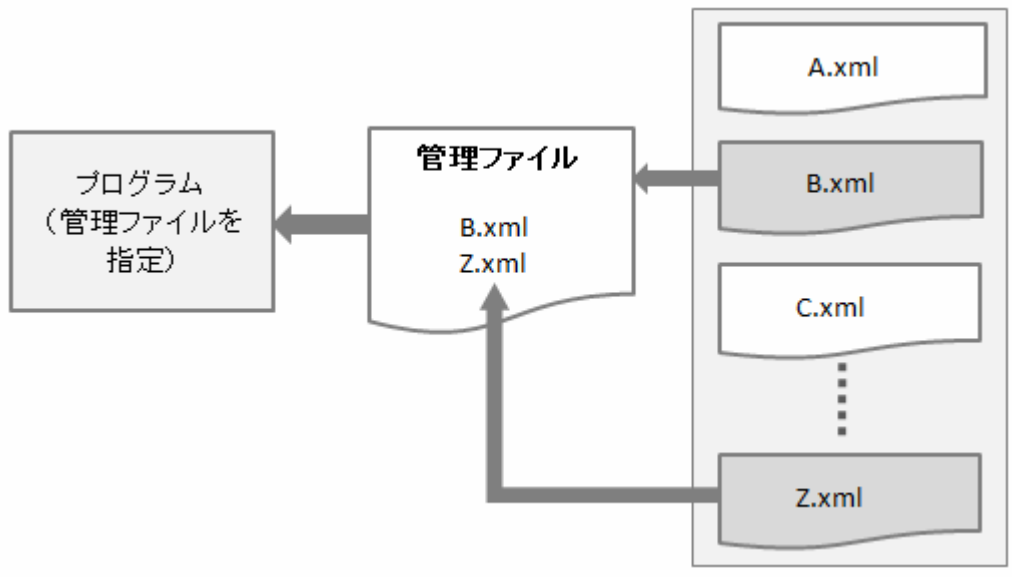
●図 5-15 複数文書ファイルの例



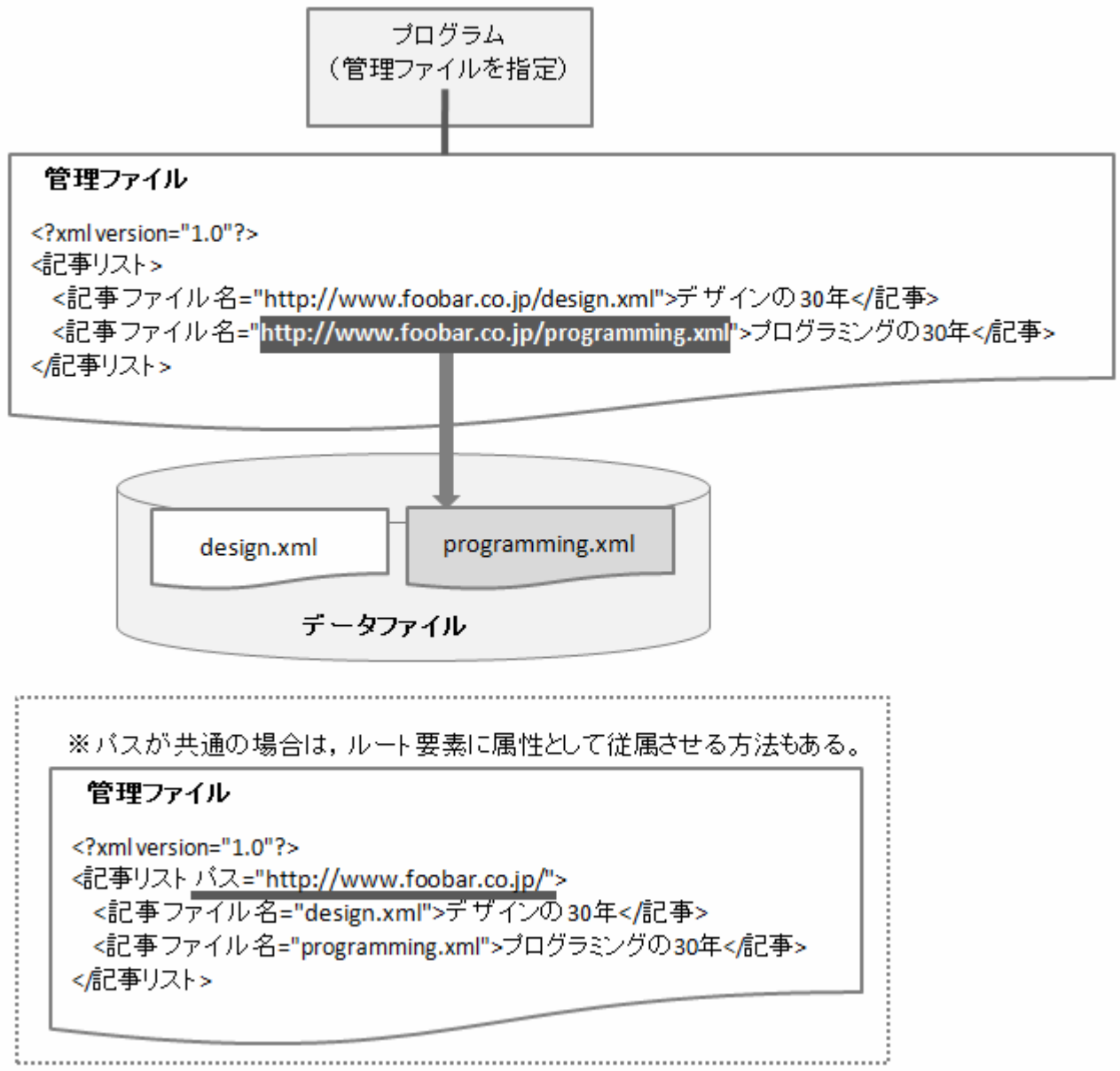
● **図 5-16** 管理ファイルを併用せず，複数のデータファイルを特定する仕組み



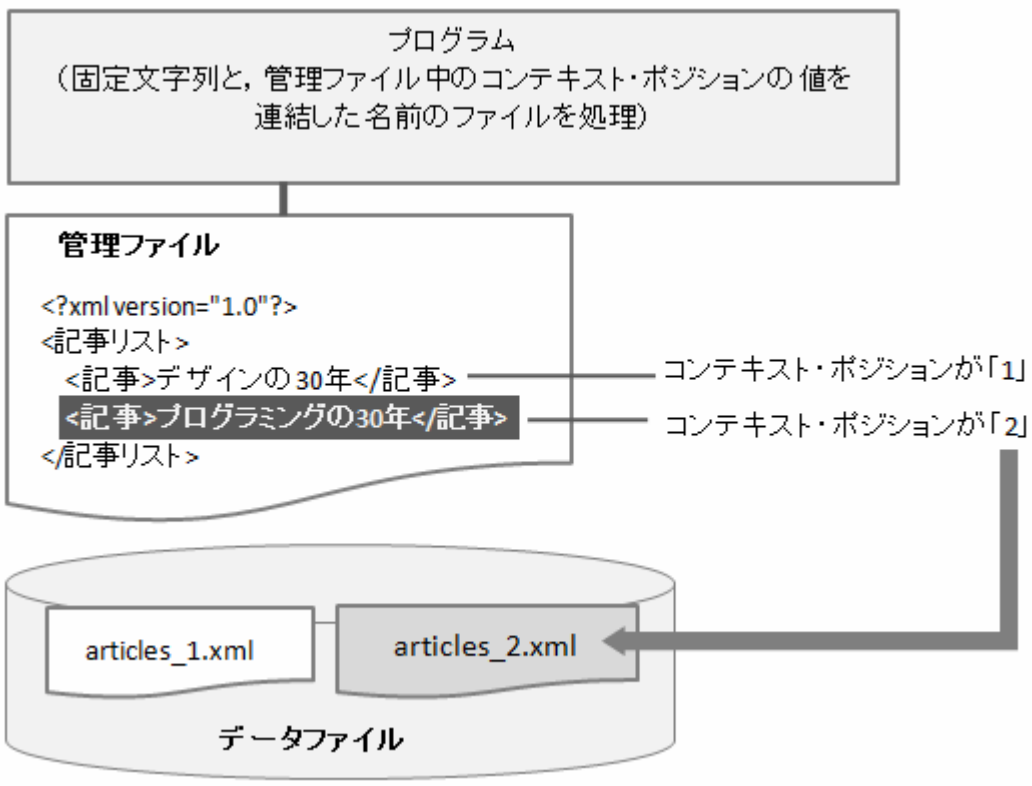
●**図 5-17** 管理ファイルを用いると，全ファイル走査ではなく，ショートカットが可能になる。



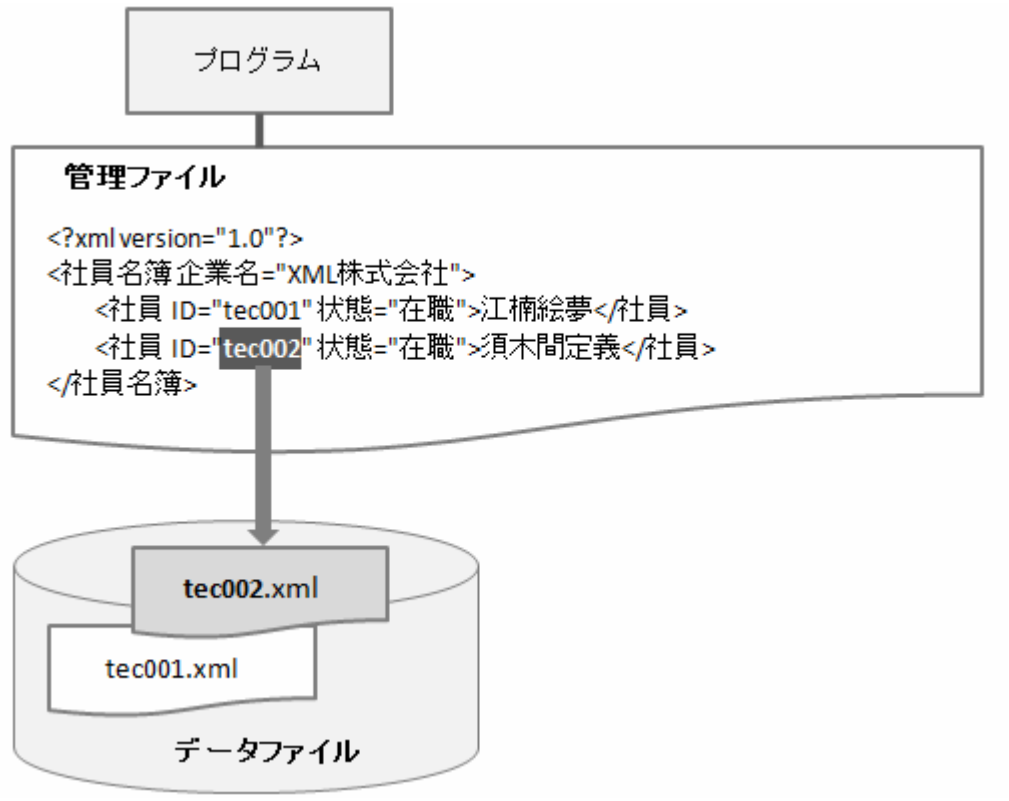
●図 5-18 処理対象データファイルの URI を，管理ファイルに記録する



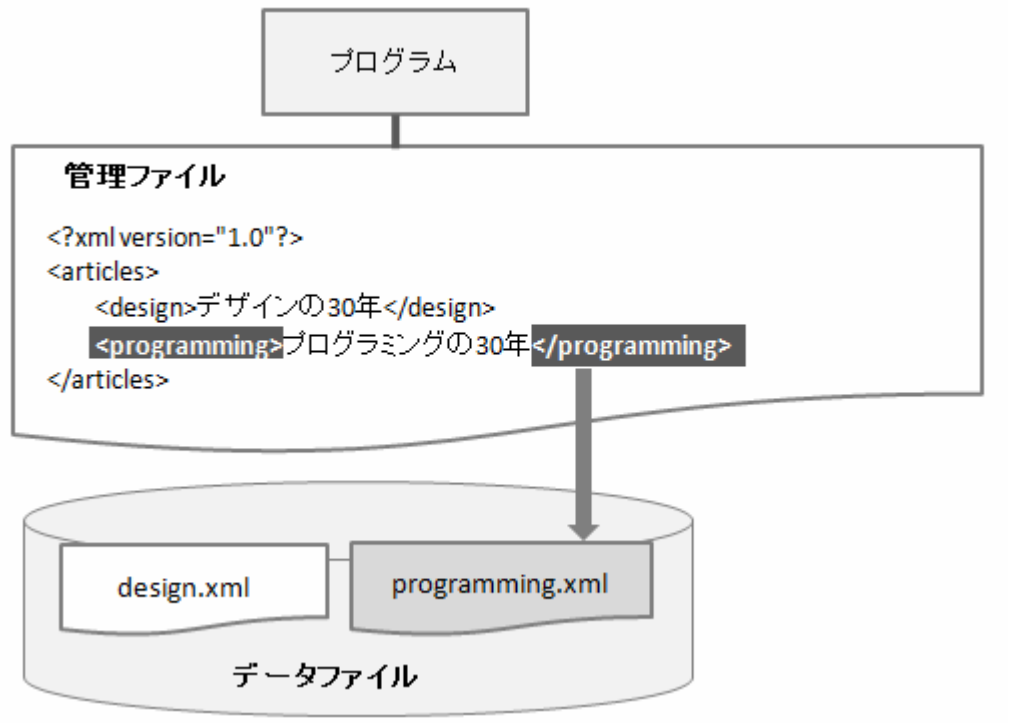
●図 5-19 各データファイルに固定文字列と連番を連結したファイル名を付ける



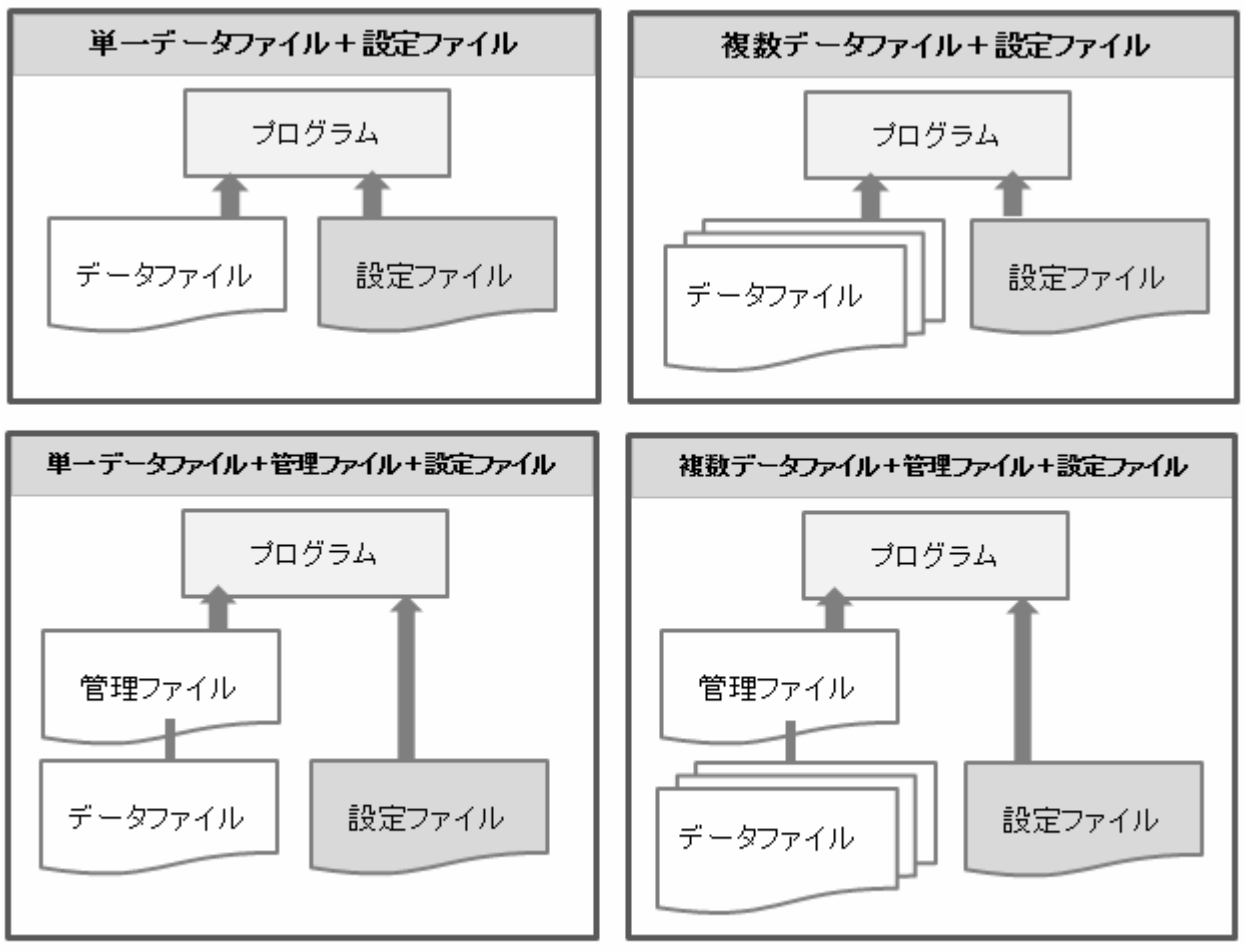
●図 5-20 管理ファイル中の要素の内容や属性の値を，データファイル名とする



●図 5-21 管理ファイル中の要素名あるいは属性名を，データファイル名とする

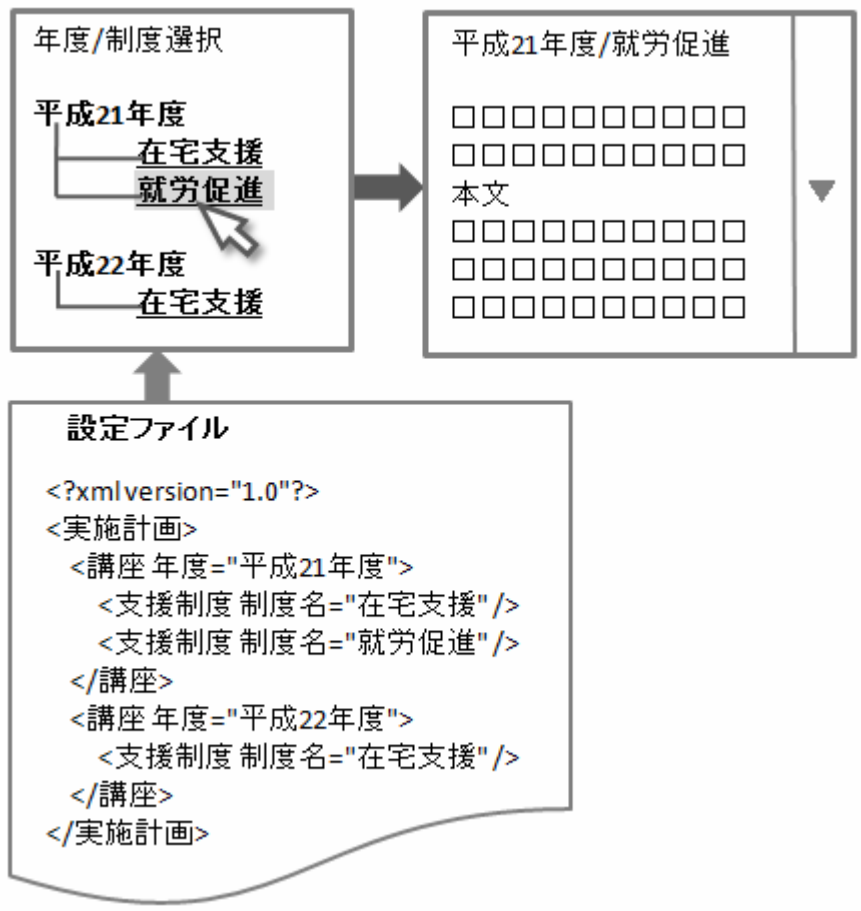


●図 5-22 設定ファイルを用いた，複数ファイルによる構成

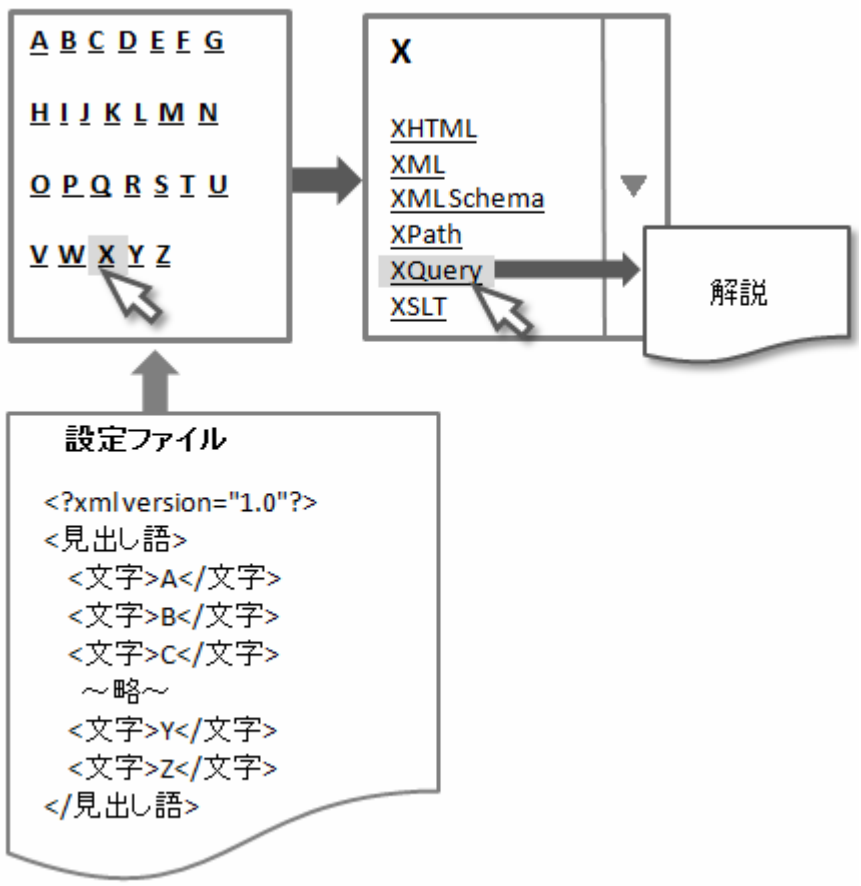


※レガシーXMLデータを記録したXML文書ファイルが1個であっても，設定ファイルがXML形式であれば，XMLアプリケーションとしては，複数XML文書ファイルによる構成となる。

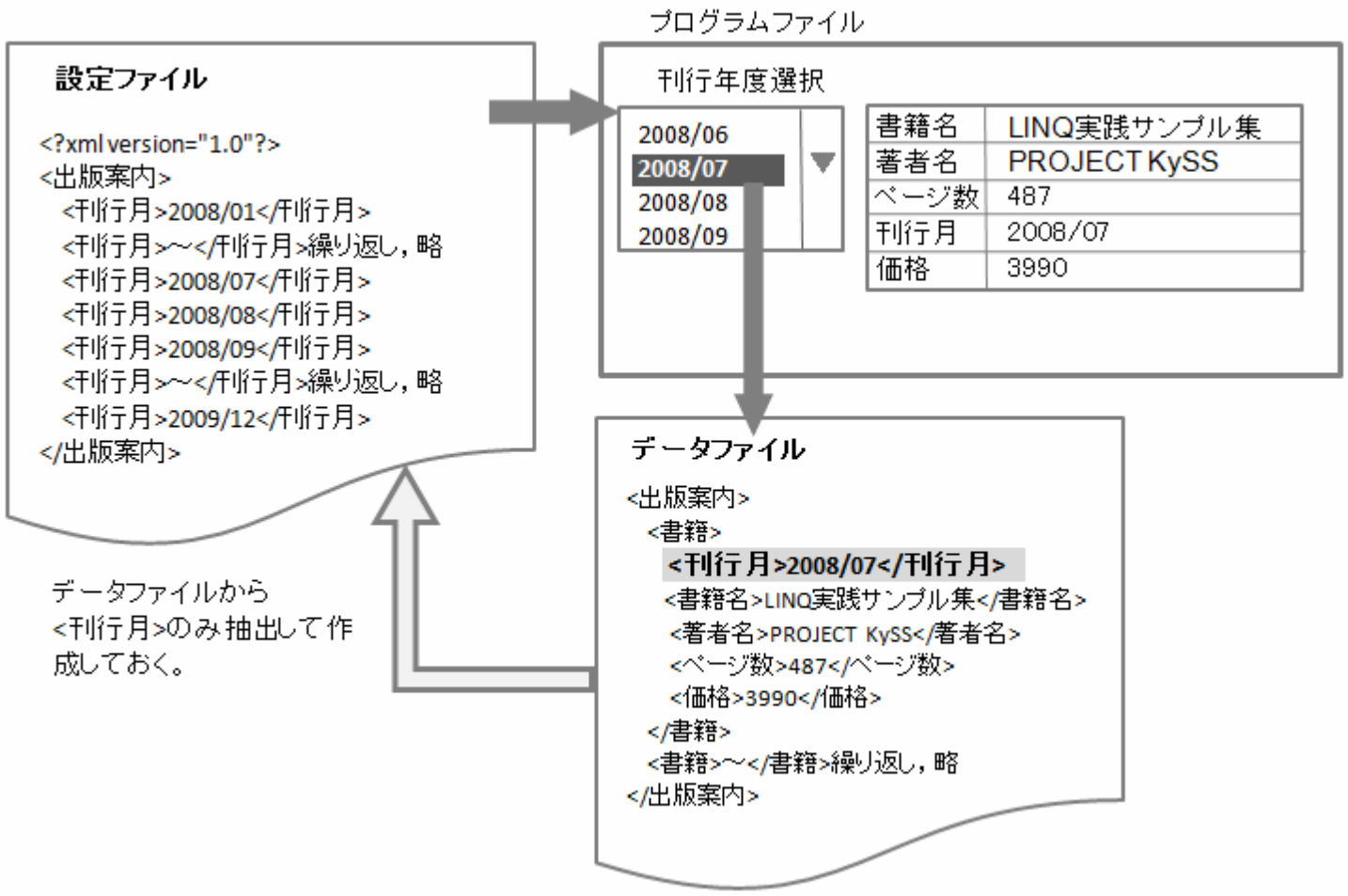
●図 5-23 設定ファイルを用いて，木構造のナビゲーションを表現する



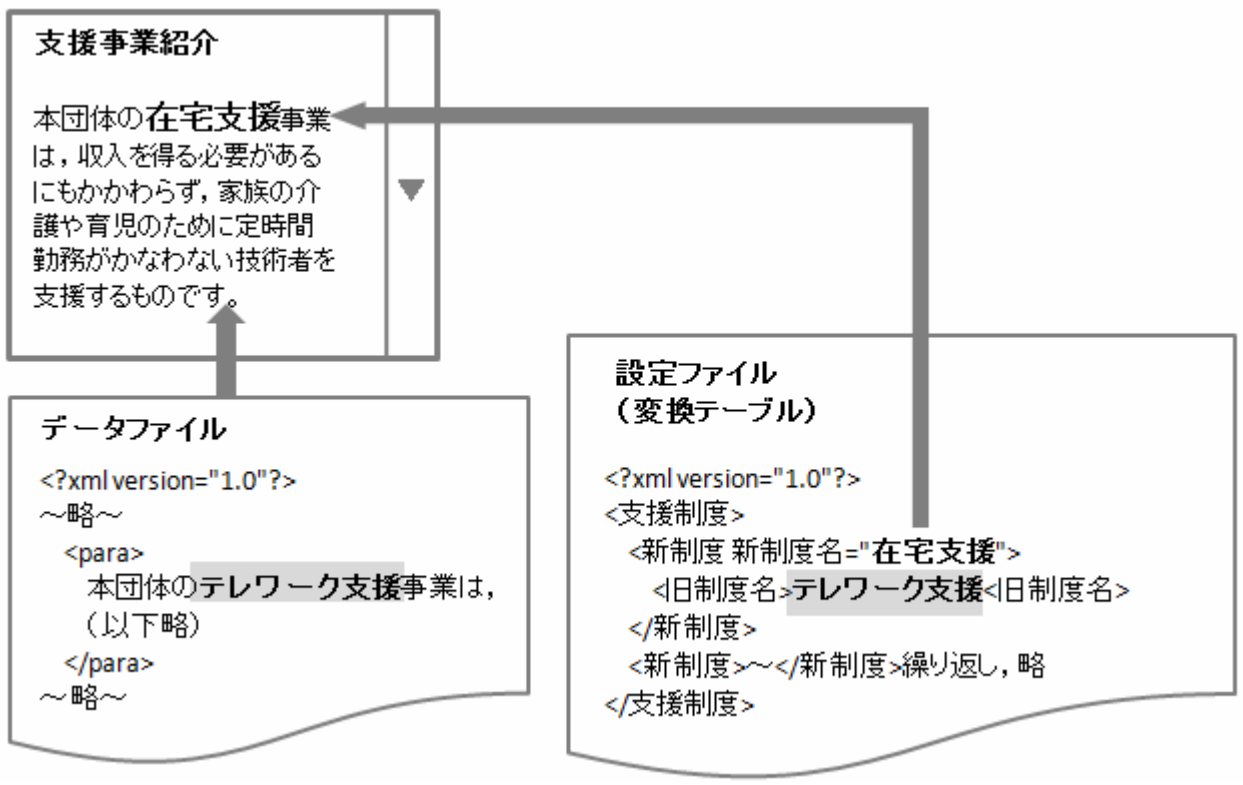
●図 5-24 見出し語選択に，設定ファイルを利用する。



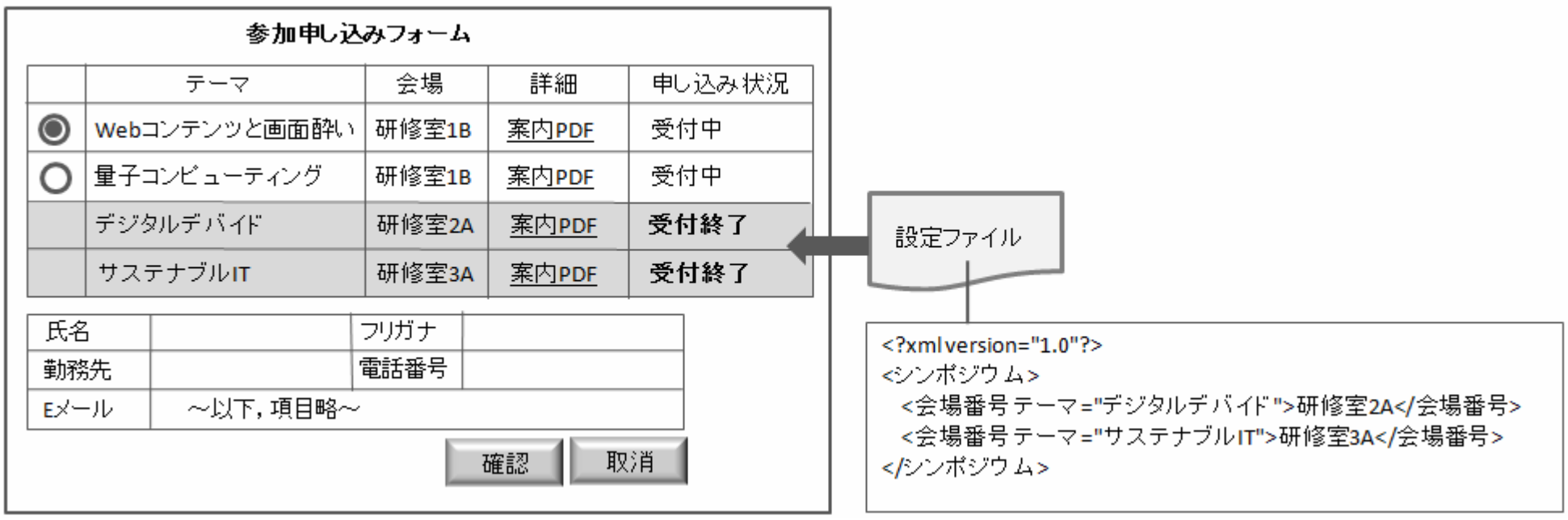
●図 5-25 データファイルのノード数が多ければ，表引き用のリストは設定ファイルとして別個に作成する



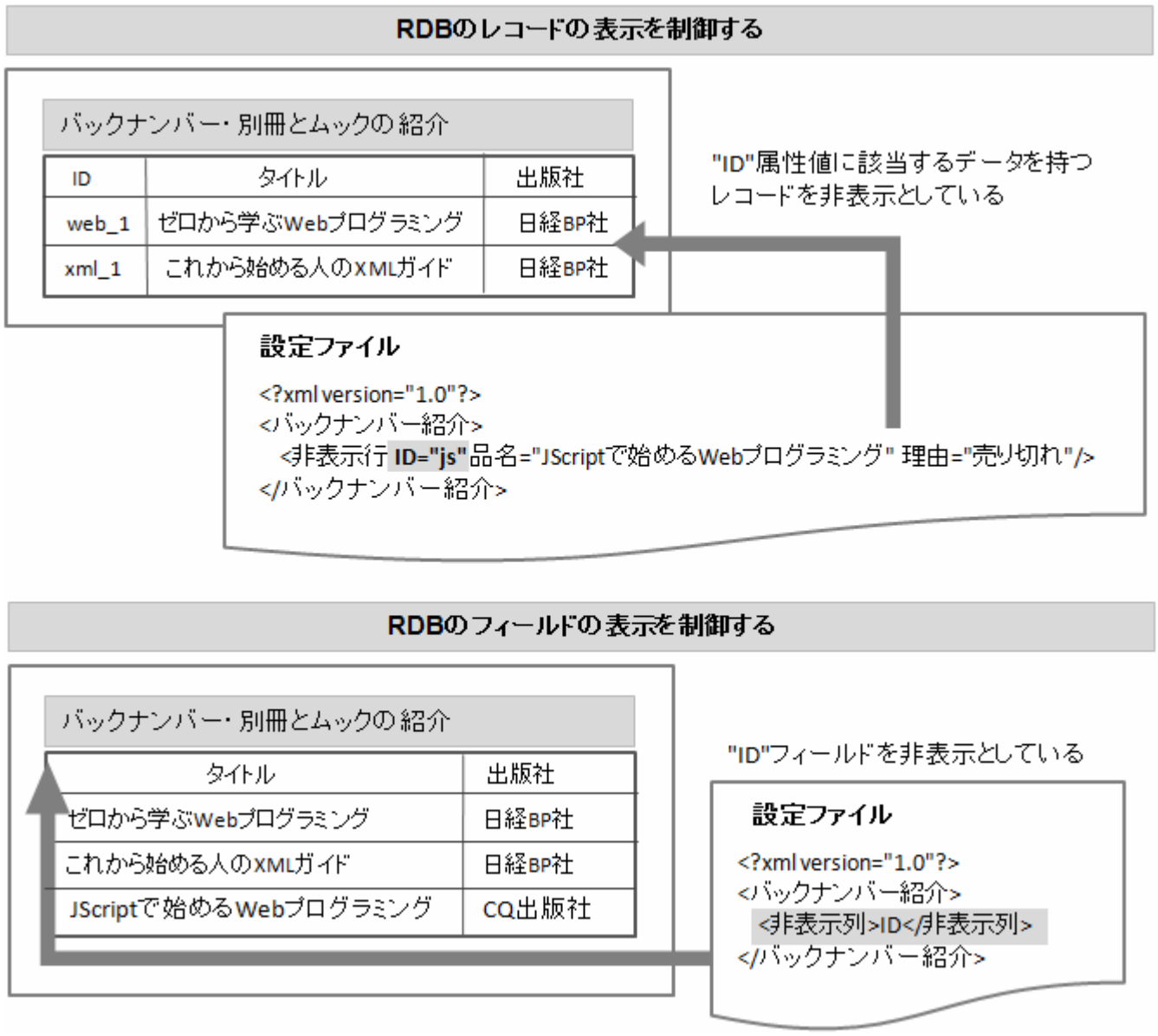
●図 5-26 設定ファイルに変換テーブルのような役割を持たせる例



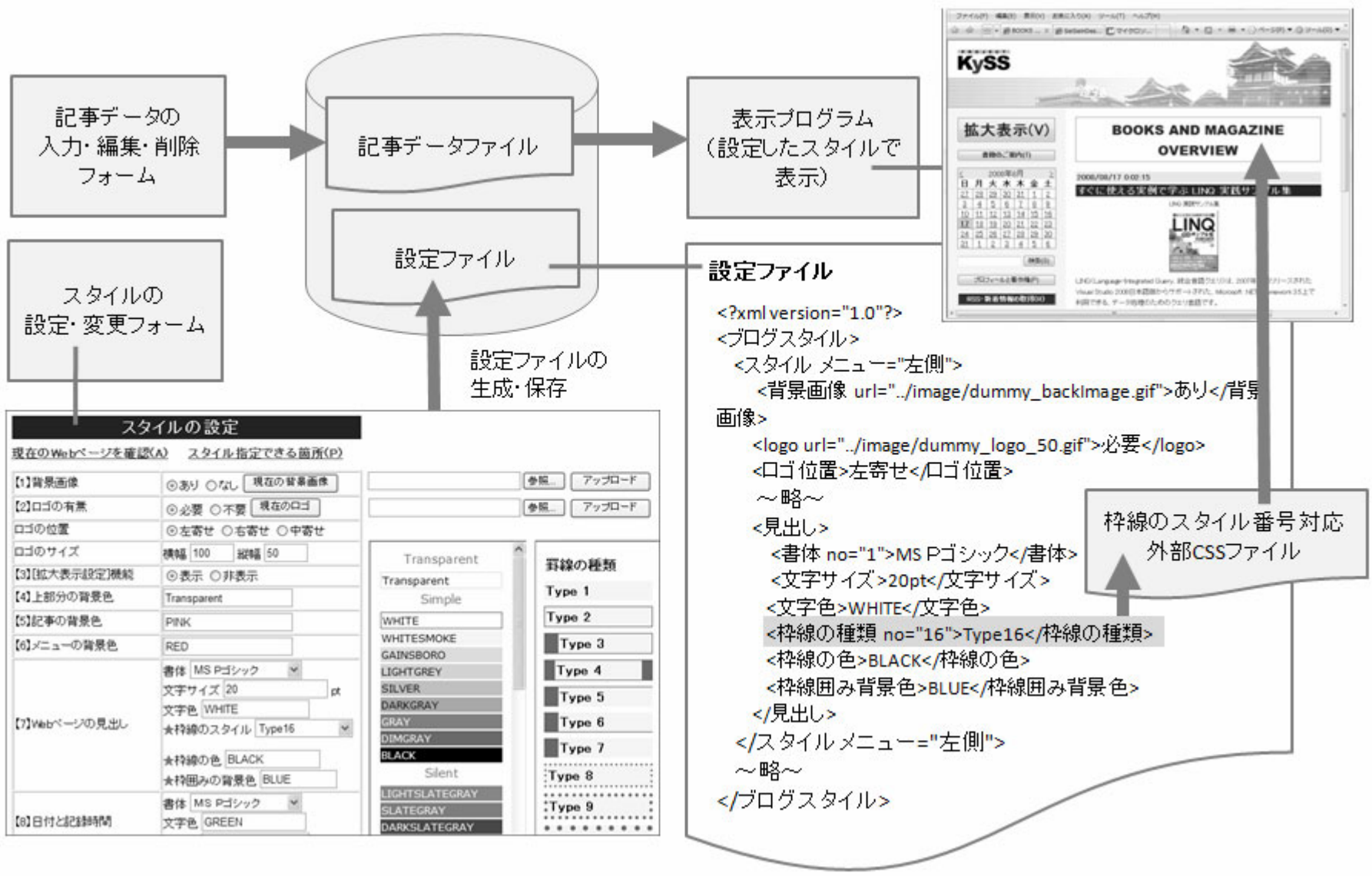
●図 5-27 設定ファイルにデータを手動で追加して，表示に反映させる例



●図 5-28 XML 設定ファイルにより，RDB の表示を制御する例



●図 5-29 ブログツールにおける XML 設定ファイルの利用例



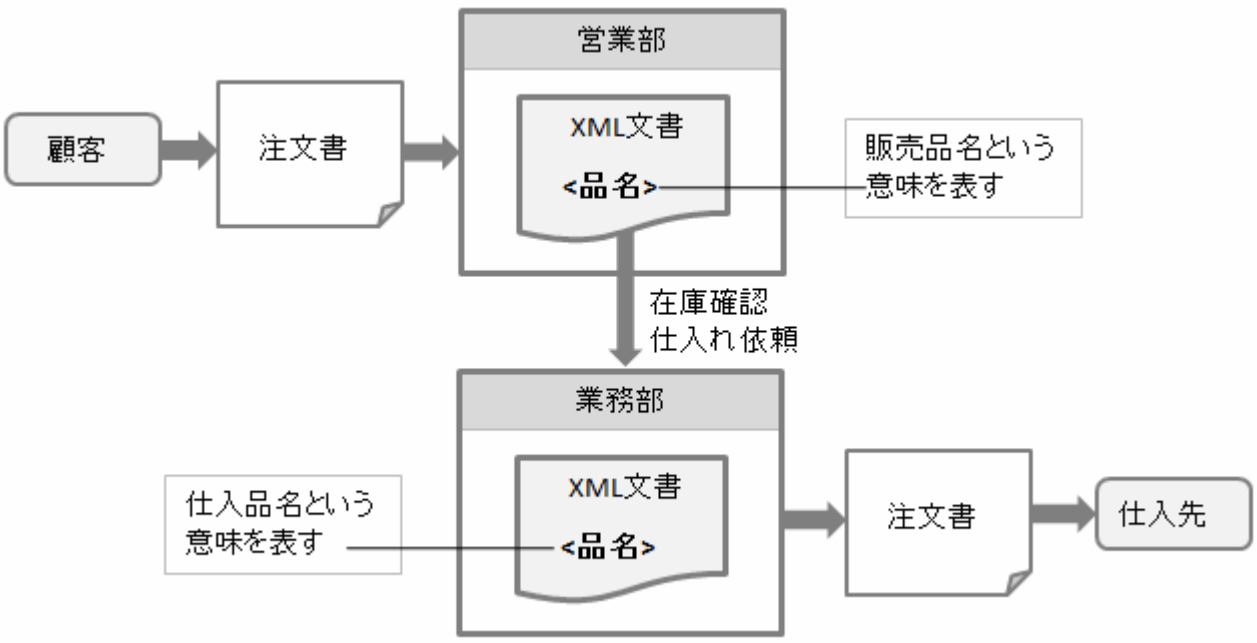
●表 5-8 管理ファイルおよびデータファイルの処理

処理	処理内容	
	管理ファイル	データファイル
変更	参照先データを変更する。	保存先やファイル名を変更する。
	参照するノードの順序を入れ替える。	必要であれば，ファイル名の連番部分を変更する。
生成・追加	末尾に参照先データを追加する。	新規ファイルを生成する。
	任意の位置に参照先データを追加する。	新規ファイルを生成する。
削除	管理ファイルを削除する。	データファイルを削除する。
	管理ファイルを削除する。	データファイルは維持する。
	管理ファイルは維持する。	データファイルを削除する。
	管理ファイルから，参照先データを削除する。	参照先のデータファイルを削除する。
	管理ファイルから，参照先データを削除する。	データファイルは維持する。
	管理ファイル中の参照先データは維持する。	参照先のデータファイルを削除する。

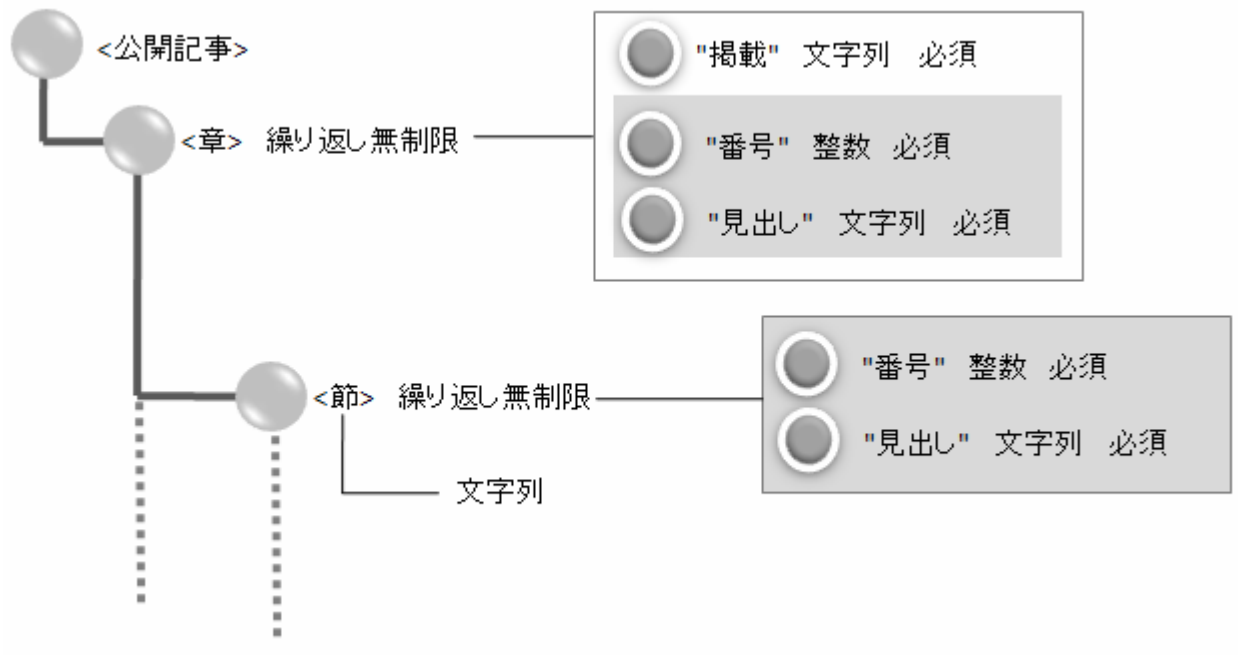
●表 5-9 単一ファイル，複数ファイル構成の特長

	単一ファイル	複数ファイル
RDBとの連携	ノード単位で連携する。主に定型繰り返しのデータと対応させることになるため，RDBとの連携の意義は薄いことがある。	ファイル単位で連携する。各ファイルの構造は不定形でもよく，完全に異なってもよい。
処理速度	特に処理にDOMを用いる場合，巨大なツリーを展開すると，初回読み込み時の処理速度がメモリに依存する	管理ファイルの設計が適切で，必要なファイルだけを読み込むのであれば，高速に処理することができる。
ノード数	最大同時アクセス数や処理の内容を考慮しないならば，（DOMを用いた処理の場合）数GBのメモリで100万ノード程度を上限とする。	ファイル単位の処理であれば，特に上限を設定する必要はない。一千万単位以上のファイル数でも，所要時間を無視するならば処理自体は可能である。
ファイル操作	ファイルのオープン/クローズが一度だけで済む。n個のファイルのオープン/走査/クローズを繰り返すのではなく，1個のファイルをオープンし，n件以下のデータを走査してクローズすることができる。	管理ファイルなしで多数のファイルを何度も読み込む処理には適していない。管理ファイルを用いた場合でも，処理によっては，ファイル数分のオープン/クローズが発生することもある。
設計	比較的容易である。	階層が深い不定形文書については，分割設計が難しい。アプリケーションの設計においても，連携処理が複雑になりがちである。
作成	オペレータのスキルによるところが大きい。階層が深く不定形であると，ミスが生じやすい。別途，XMLデータ生成アプリケーションを開発する方法もある。	簡易な構造であれば，作成の分業が可能である。単一ファイルを分割する，XMLデータ生成アプリケーションを開発する方法もある。
管理（更新方法）	管理するファイルが1個でよいので，一人の担当者が管理する場合は，データの更新作業やファイルの管理が容易である。	ファイルが適切に分かれていれば，変更の必要なファイルのみ更新すればよいため，メンテナンスが容易である。上書き問題も回避しやすい。他方で，データ更新を自動化せず手作業で行う場合，管理ファイル側の関連データの修正漏れが発生する恐れがある。

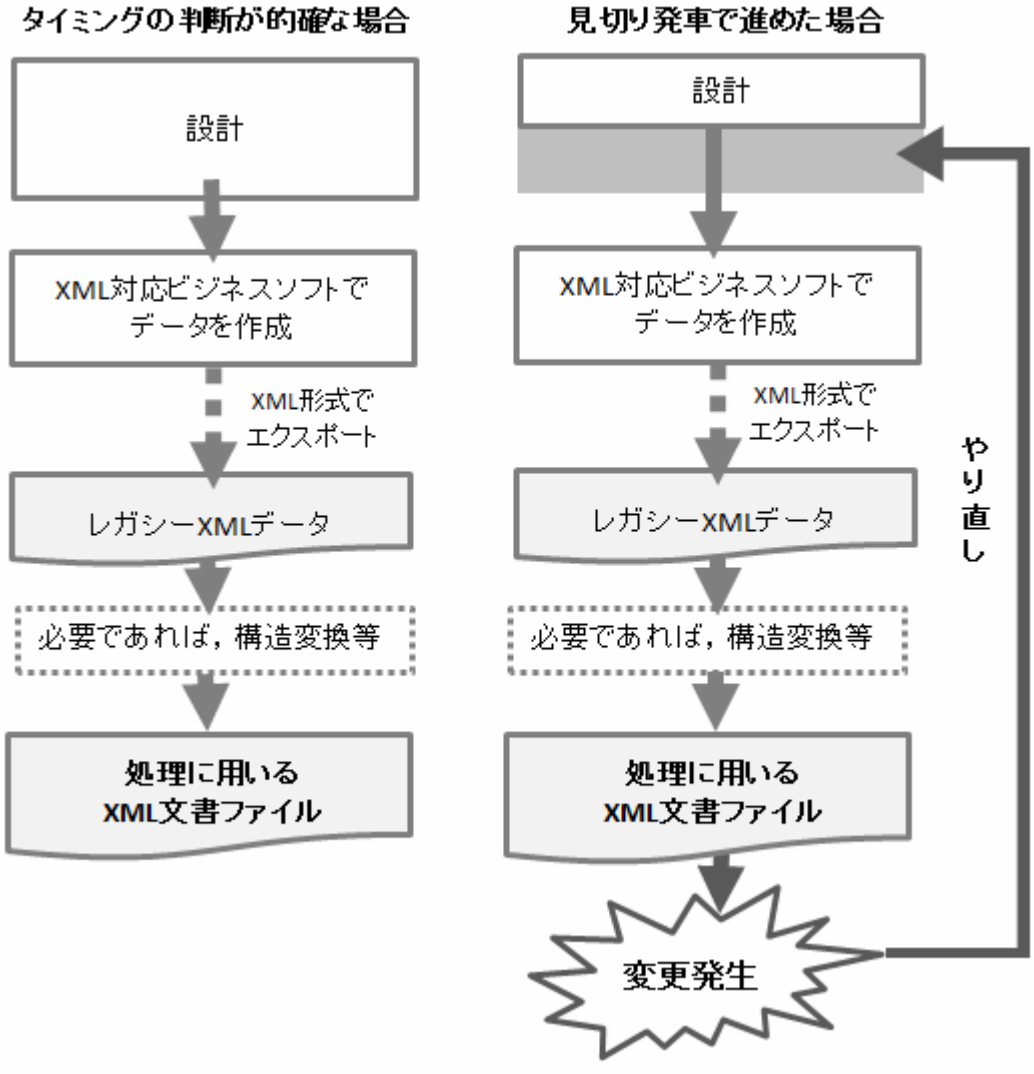
●図 5-30 同じ要素名であっても，文書利用者の立場によって意味が異なることがある。



●図 5-31 技術者以外に伝えるには，木構造を絵で表現すると理解してもらいやすい。

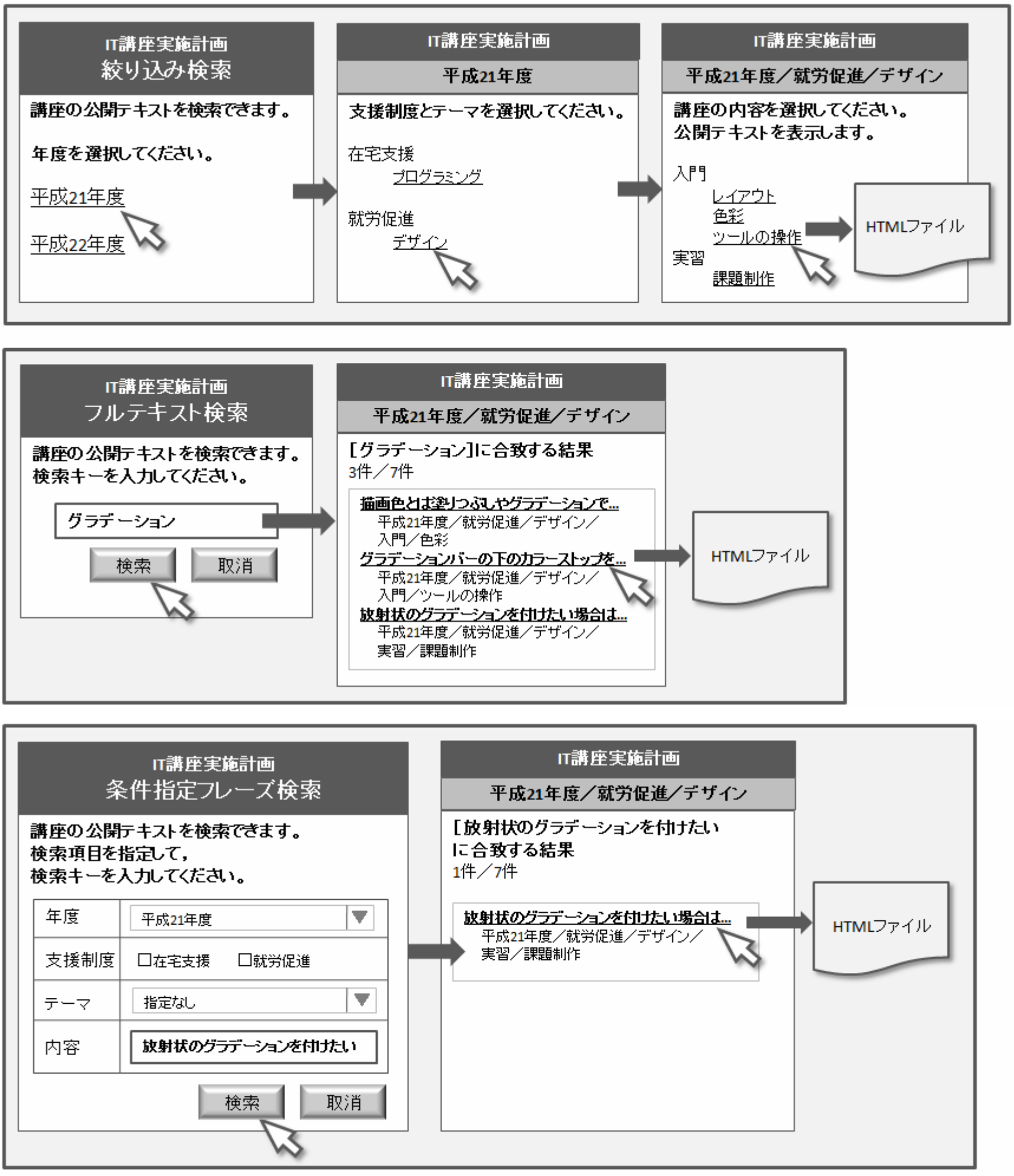


●図 5-32 XML 化のタイミングを正しく判断し，手間とミスを防ぐ



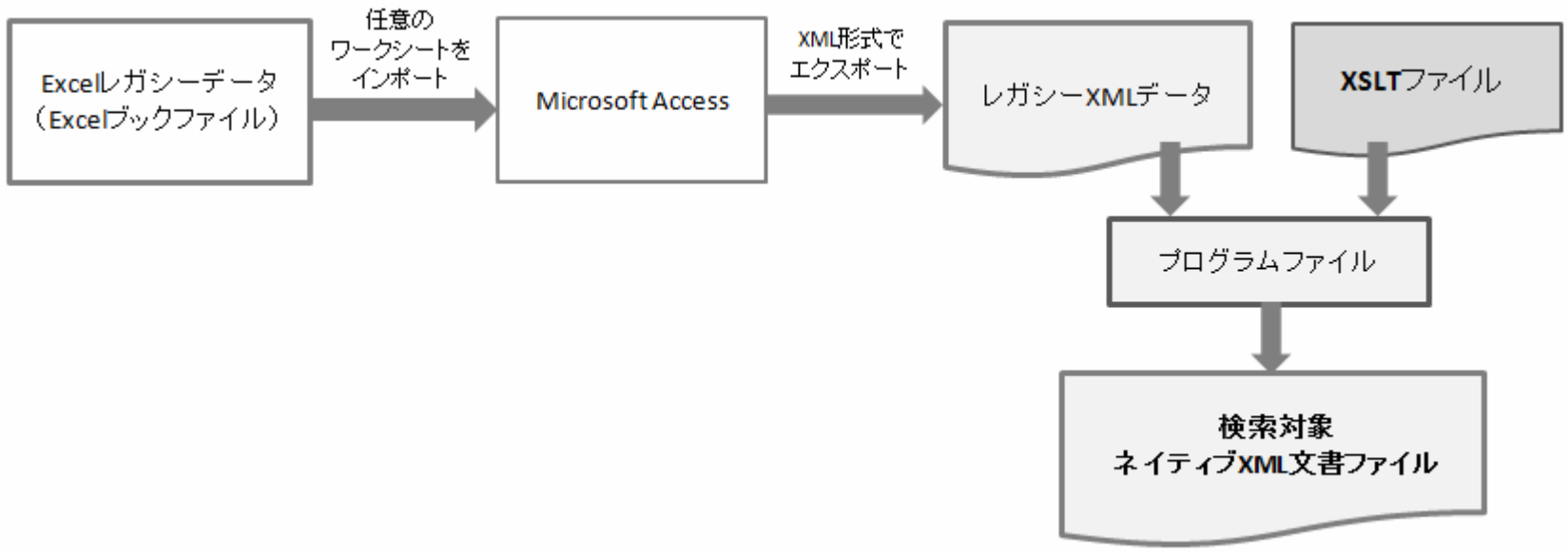
第 6 章の図表

●図 6-1 XML の特長を生かした検索処理。絞り込み検索，フルテキスト検索，条件指定フレーズ検索。



※この図は，Webアプリケーションとして実装した場合の画面遷移例である。
※図を分かりやすくするため，通常，検索アプリケーション画面にある，検索手順のヘルプ，1ページあたりの表示件数，ページング等は記載していない。

●図 6-2 Excel ワークシートから，検索処理対象とするネイティブ XML 文書を作成する，一方法。



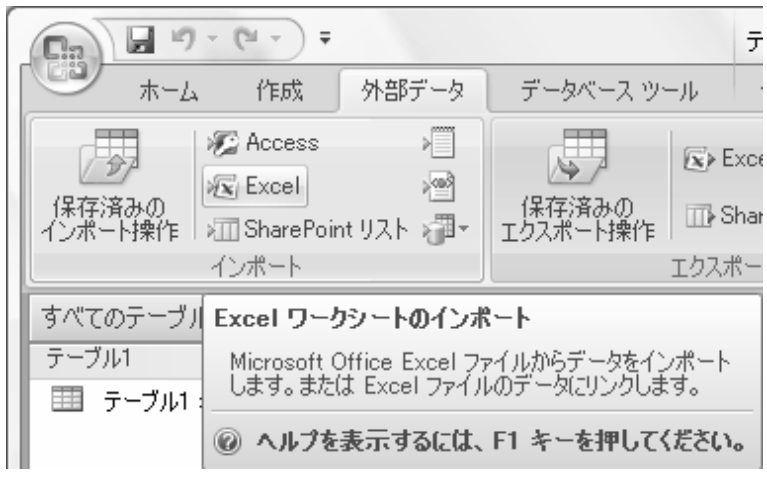
Microsoft Excel で入力されたデータ

	A	B	C	D	E	F	G	H	I	J	K
1	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索テキスト	所属1	氏名1	所属2
2	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_01.xml	女性技術者の会	古見　久美	
3	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_02.xml	シルバーギーク会	秋葉　定数	
4	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_03.xml	シルバーギーク会	木板　雄拓	女性技術者の
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	design_01.xml	XML株式会社	江楠　絵夢	
6	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	design_02.xml	XML株式会社	江楠　絵夢	
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	design_03.xml	XML株式会社	江楠　絵夢	夢織デザイン
8	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	design_04.xml	XML株式会社	江楠　絵夢	視覚スタジオ
9	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_01.xml	—	—	
10	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_02.xml	XML株式会社	須木間　定義 江楠　絵夢	DBユーザグル
11	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_03.xml	XML株式会社	須木間　定義 江楠　絵夢	視覚スタジオ
12											

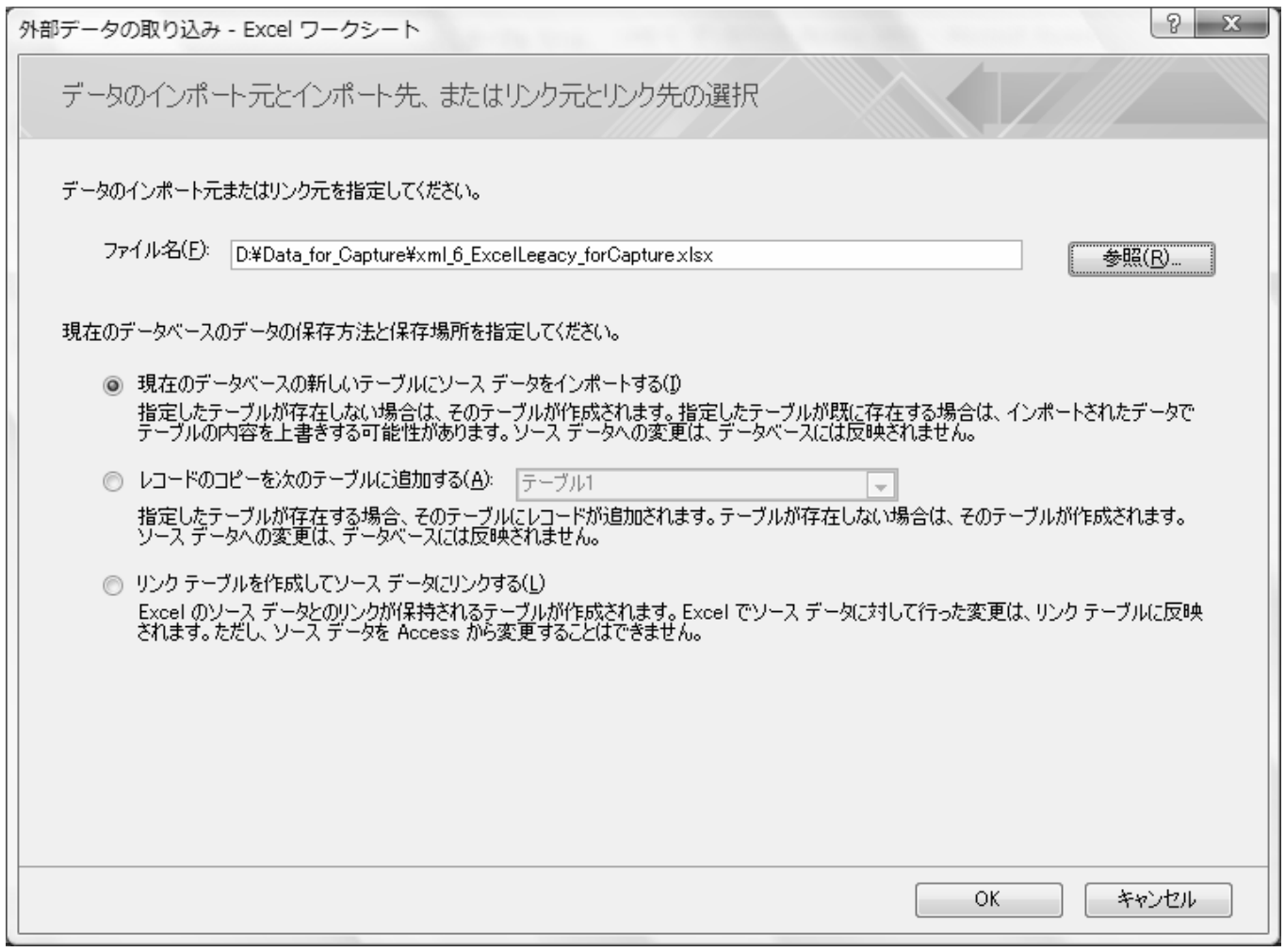
Microsoft Access の操作手順

①空のデータベースを新規作成。

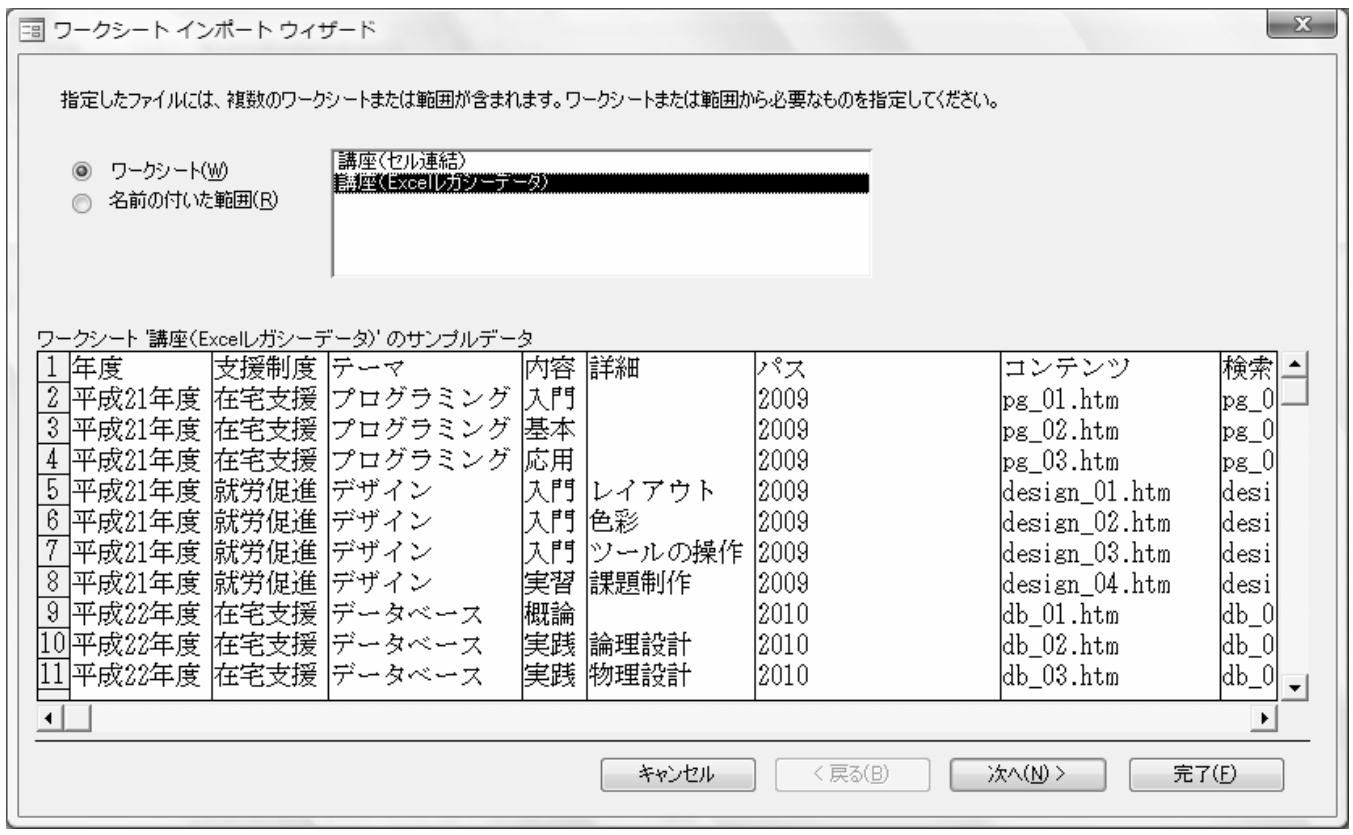
②［外部データ］タブ［Excel］選択。



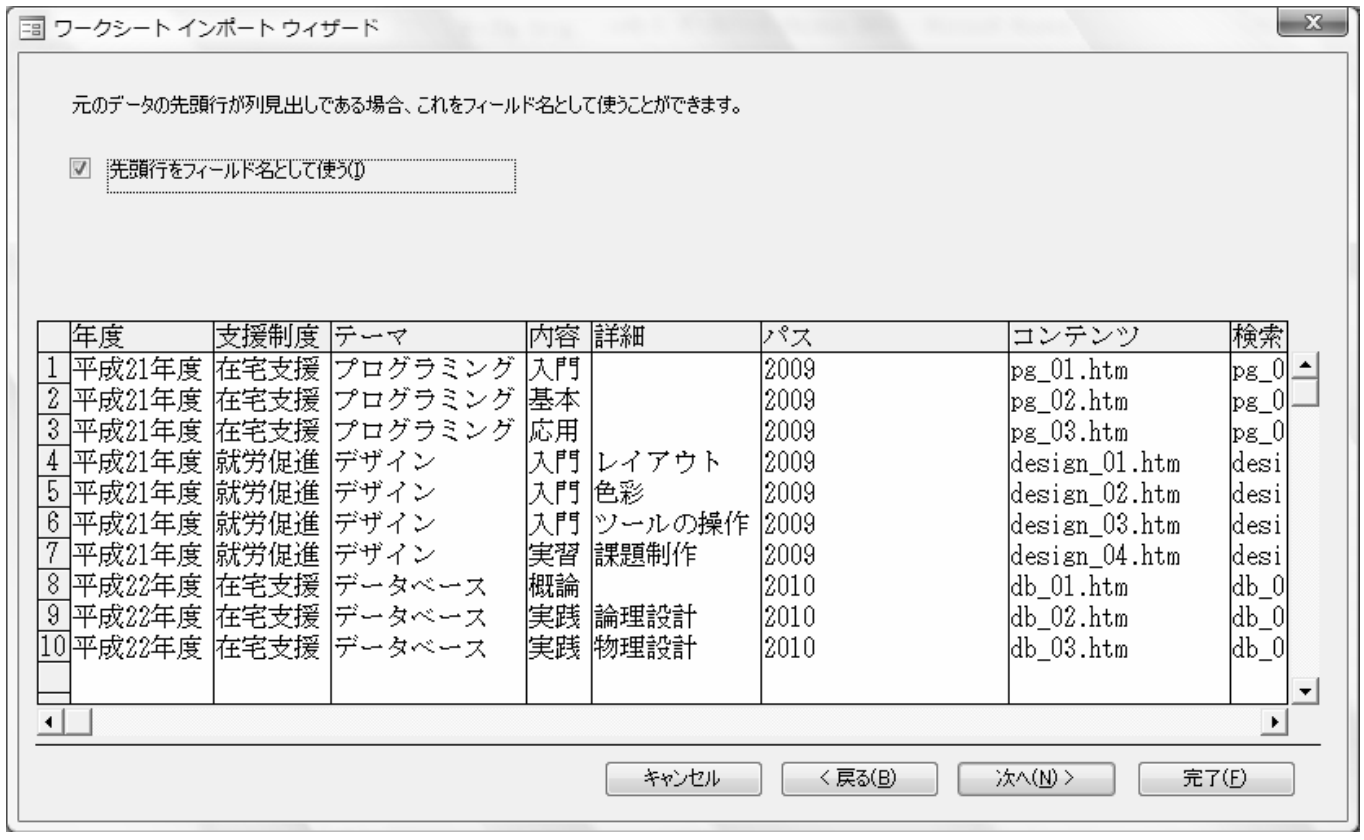
③［参照］クリック，インポートする Excel ファイルを指定して［OK］。



④ワークシート選択，［次へ］。

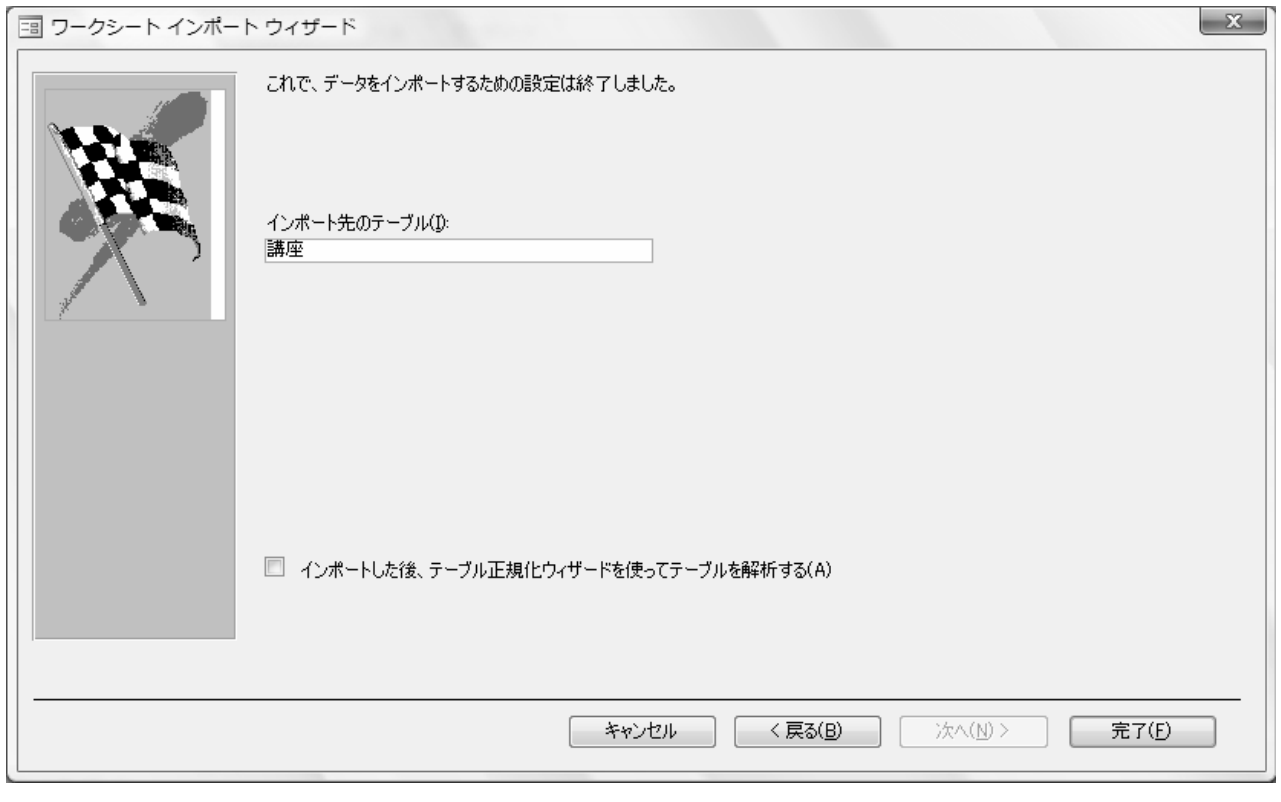


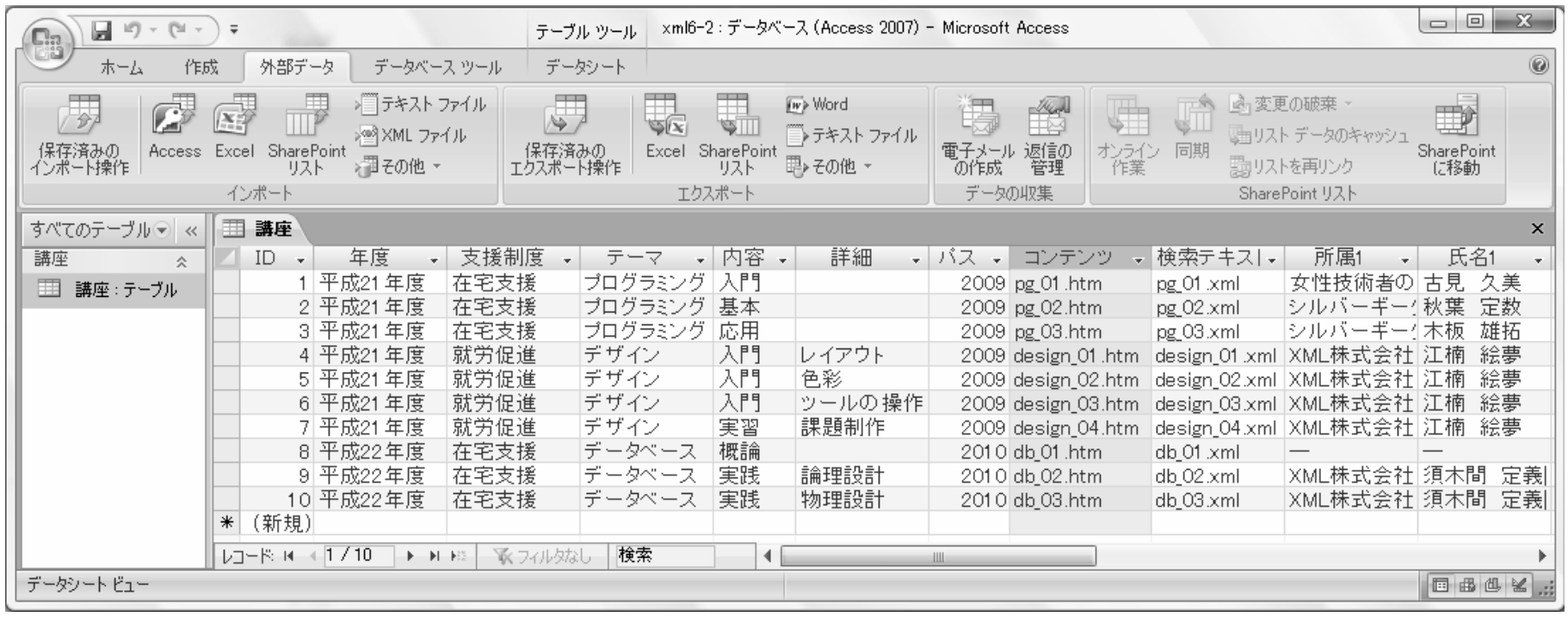
⑤［先頭行をフィールド名として使う(I)］にチェックを付けて，［次へ］さらに［次へ］。



⑥［主キーを自動的に設定する(A)］にチェックがデフォルトの状態で，［次へ］。

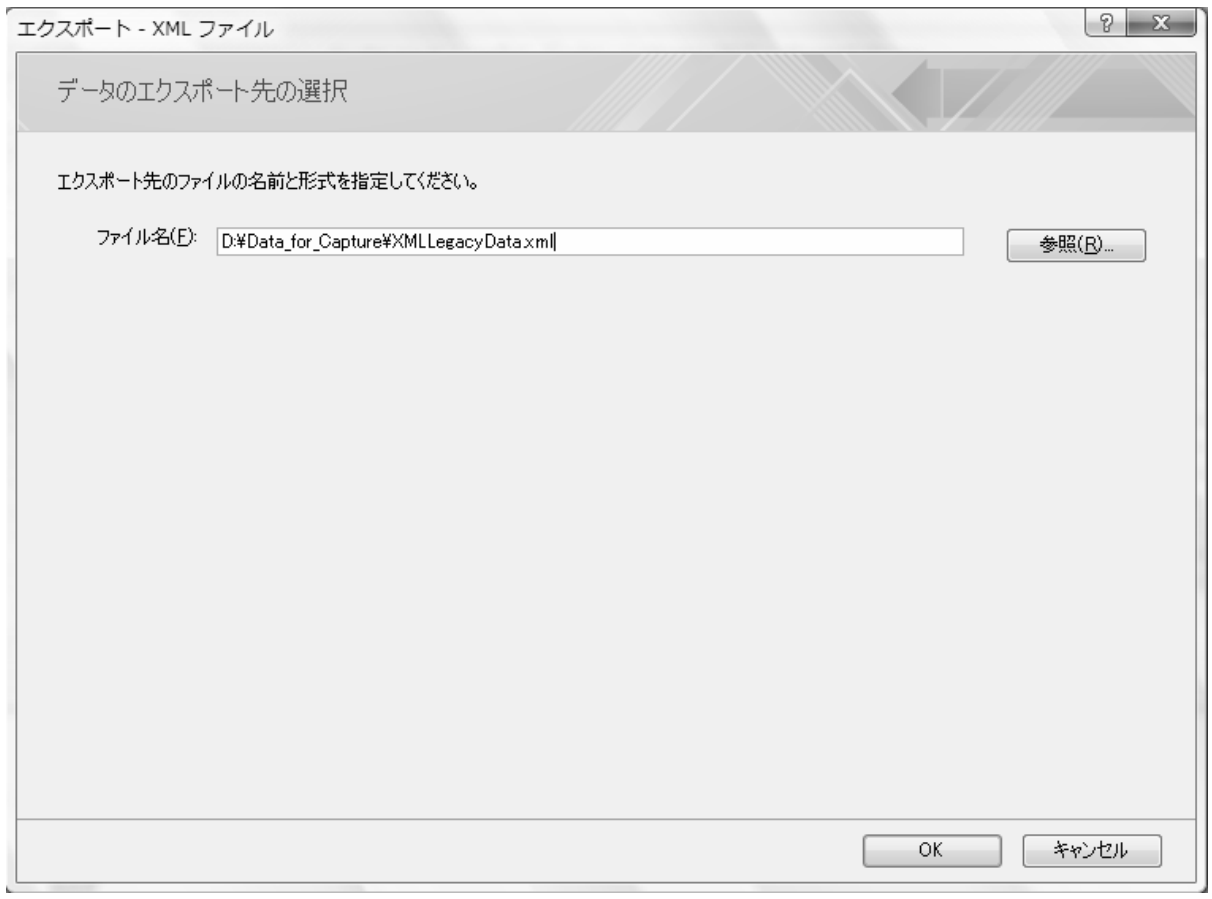
⑦［インポート先のテーブル名］を入力して，［完了］［閉じる］。このテーブル名が，XML 形式でエクスポートした際に，ルート要素直下で繰り返す要素の名前となる。





⑧［エクスポート］［その他］［XML ファイル］選択。

⑨エクスポート先と XML ファイル名を指定。



⑩［データ (XML) (D)］のみにチェックして［OK］。（デフォルトでは［データのスキーマ (XSD) (S)］にもチェックが付いている），［閉じる］。

●図 6-3 XML 化は考慮せず入力された一般的な Excel ワークシート。

	A	B	C	D	E	F	G	H
1	年度	支援制度	テーマ	内容		コンテンツ	講師	
2							所属	氏名
3	平成21年度	在宅支援	プログラミング	入門		2009/pg_01.htm	女性技術者の会	古見　久美
4				基本		2009/pg_02.htm	シルバーギーク会	秋葉　定数
5				応用		2009/pg_03.htm	シルバーギーク会	木板　雄拓
6							女性技術者の会	半田　富士子，半田　妹子
7		就労促進	デザイン	入門	レイアウト	2009/design_01.htm	XML株式会社	江楠　絵夢
8					色彩	2009/design_02.htm	XML株式会社	江楠　絵夢
9					ツールの操作	2009/design_03.htm	XML株式会社	江楠　絵夢
10							夢織デザイン	阿戸　美子
11				実習	課題制作	2009/design_04.htm	XML株式会社	江楠　絵夢
12							視覚スタジオ	志井　紗歩
13							夢織デザイン	阿戸　美子
14	平成22年度	在宅支援	データベース	概論		2010/db_01.htm		
15				実践	論理設計	2010/db_02.htm	XML株式会社	須木間　定義，江楠　絵夢
16							DBユーザグループ	江須久　絵瑠
17					物理設計	2010/db_03.htm	XML株式会社	須木間　定義，江楠　絵夢
18							視覚スタジオ	林　功

●図 6-4 XML 化に適した Excel ワークシート。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	年度	支援制度	テーマ	内容	詳細	パス	コンテンツ	検索テキスト	所属1	氏名1	所属2	氏名2	所属3	氏名3
2	平成21年度	在宅支援	プログラミング	入門		2009	pg_01.htm	pg_01.xml	女性技術者の会	古見　久美				
3	平成21年度	在宅支援	プログラミング	基本		2009	pg_02.htm	pg_02.xml	シルバーギーク会	秋葉　定数				
4	平成21年度	在宅支援	プログラミング	応用		2009	pg_03.htm	pg_03.xml	シルバーギーク会	木板　雄拓	女性技術者の会	半田　富士子|半田　妹子		
5	平成21年度	就労促進	デザイン	入門	レイアウト	2009	design_01.htm	design_01.xml	XML株式会社	江楠　絵夢				
6	平成21年度	就労促進	デザイン	入門	色彩	2009	design_02.htm	design_02.xml	XML株式会社	江楠　絵夢				
7	平成21年度	就労促進	デザイン	入門	ツールの操作	2009	design_03.htm	design_03.xml	XML株式会社	江楠　絵夢	夢織デザイン	阿戸　美子		
8	平成21年度	就労促進	デザイン	実習	課題制作	2009	design_04.htm	design_04.xml	XML株式会社	江楠　絵夢	視覚スタジオ	志井　紗歩	夢織デザイン	阿戸　美子
9	平成22年度	在宅支援	データベース	概論		2010	db_01.htm	db_01.xml	―	―				
10	平成22年度	在宅支援	データベース	実践	論理設計	2010	db_02.htm	db_02.xml	XML株式会社	須木間　定義|江楠　絵夢	DBユーザグループ	江須久　絵瑠		
11	平成22年度	在宅支援	データベース	実践	物理設計	2010	db_03.htm	db_03.xml	XML株式会社	須木間　定義|江楠　絵夢	視覚スタジオ	林　功		

必ず1回出現

データがなくても，
空要素を生成しない

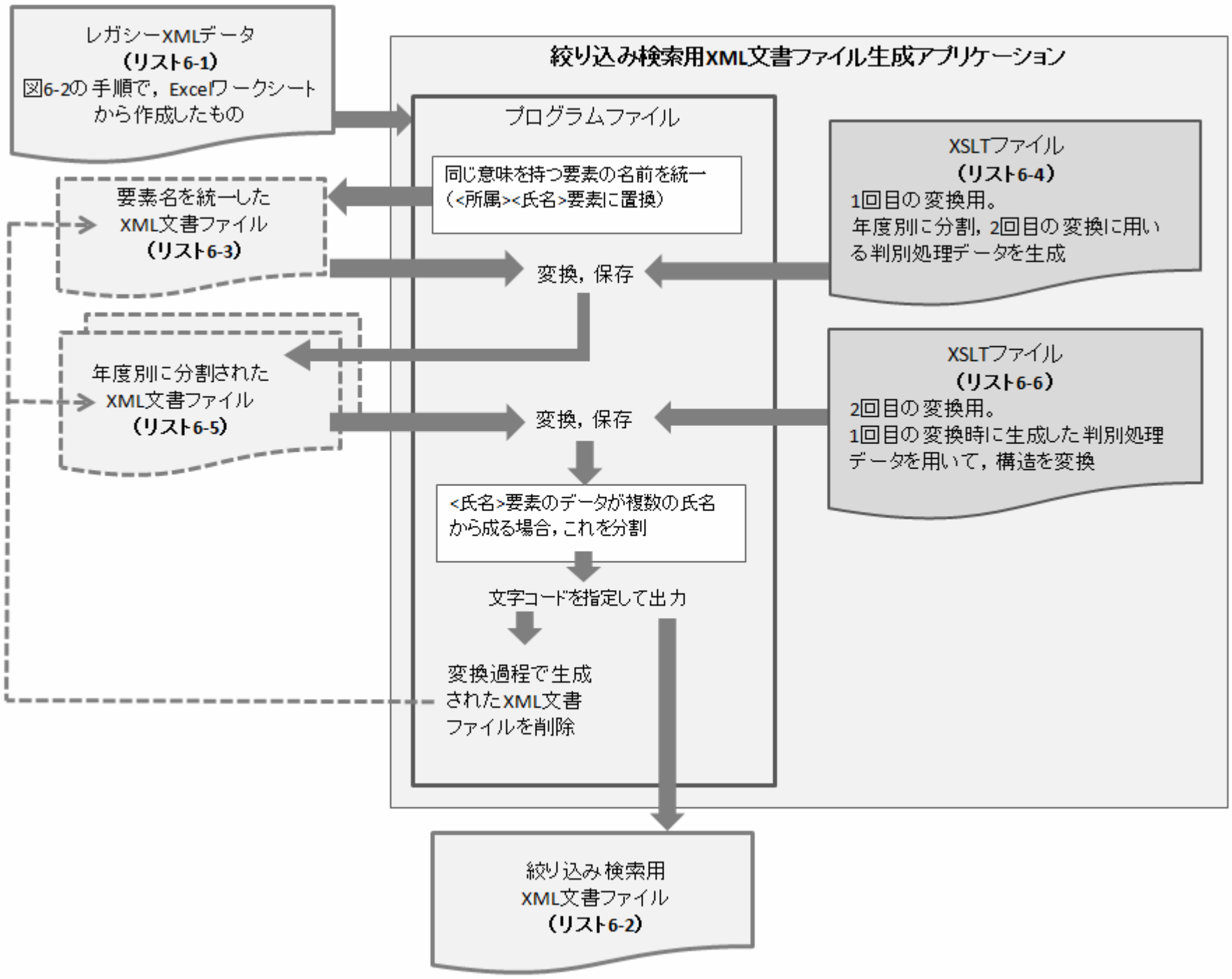
テキスト検索，
あるいはタグ名指定検索に
利用する，本文データを記録
したXMLファイル名

データがなくても，「―」を
内容とする要素を生成する

所属と講師については，さらに増える可能
性に対応するため，最後に入力している

必ず1回出現

●図 6-5 ●リスト 6-1 を●リスト 6-2 のような構造の複数文書に変換する処理の流れの例



●図 6-6 アンケート集計表などでよく見受けられる"0"や"1"が記入されている例。(図 6-6.bmp)

	A	B	C	D	E
1	平成21年度	平成22年度	在宅支援制度	就労促進制度	テーマ
2	1	0	1	0	プログラミン
3	1	0	1	0	プログラミン
4	1	0	1	0	プログラミン
5	1	0	0	1	デザイン
6	1	0	0	1	デザイン
7	1	0	0	1	デザイン
8	1	0	0	1	デザイン
9	0	1	1	0	データベース
10	0	1	1	0	データベース
11	0	1	1	0	データベース

●図 6-7 誤った構造変換を避けるために，データ重複を予測する。

現在のデータ

支援制度	テーマ	内容	詳細
在宅支援	プログラミング	入門	
在宅支援	プログラミング	基本	
在宅支援	プログラミング	応用	
就労促進	デザイン	入門	レイアウト
就労促進	デザイン	入門	色彩
就労促進	デザイン	入門	ツールの操作
就労促進	デザイン	実習	課題制作

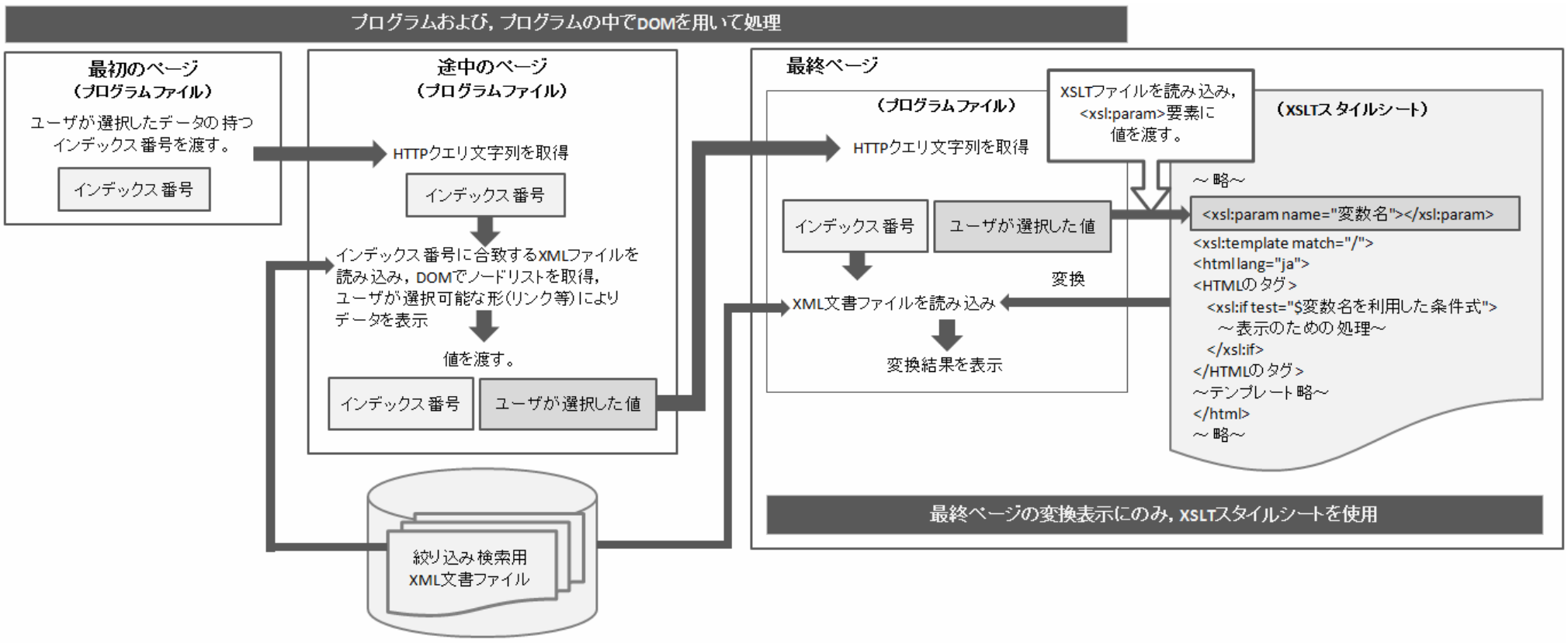
予測されるデータの例

支援制度	テーマ	内容	詳細
在宅支援	プログラミング	入門	ツールの操作
在宅支援	プログラミング	基本	
在宅支援	プログラミング	応用	
在宅支援	デザイン	入門	ツールの操作
就労促進	デザイン	入門	レイアウト
就労促進	デザイン	入門	色彩
就労促進	デザイン	入門	ツールの操作
就労促進	デザイン	実習	課題制作

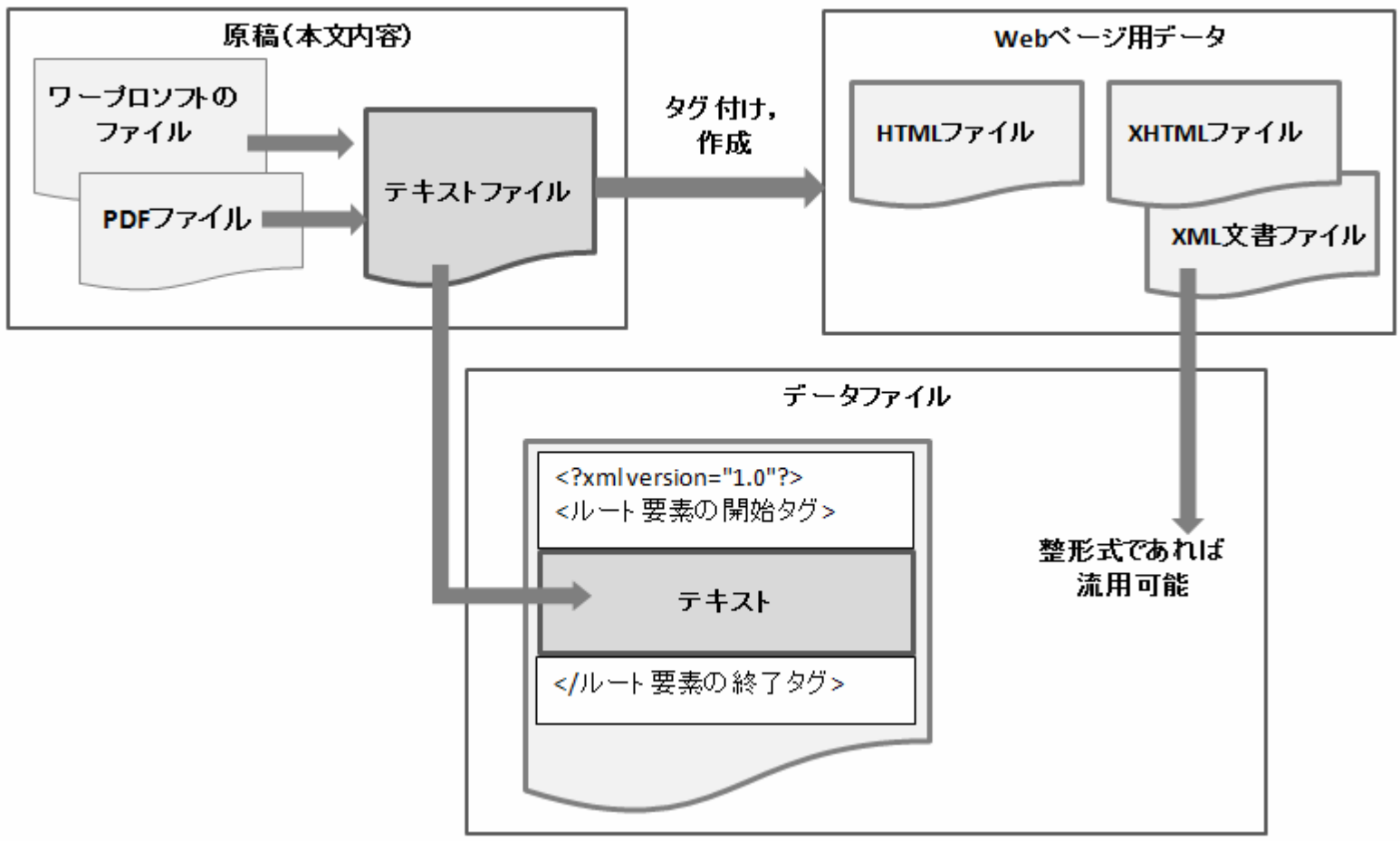
追加されたデータ

「<テーマ>が重複していなければ」という条件だけでは，構造変換の際にスルーされてしまう。

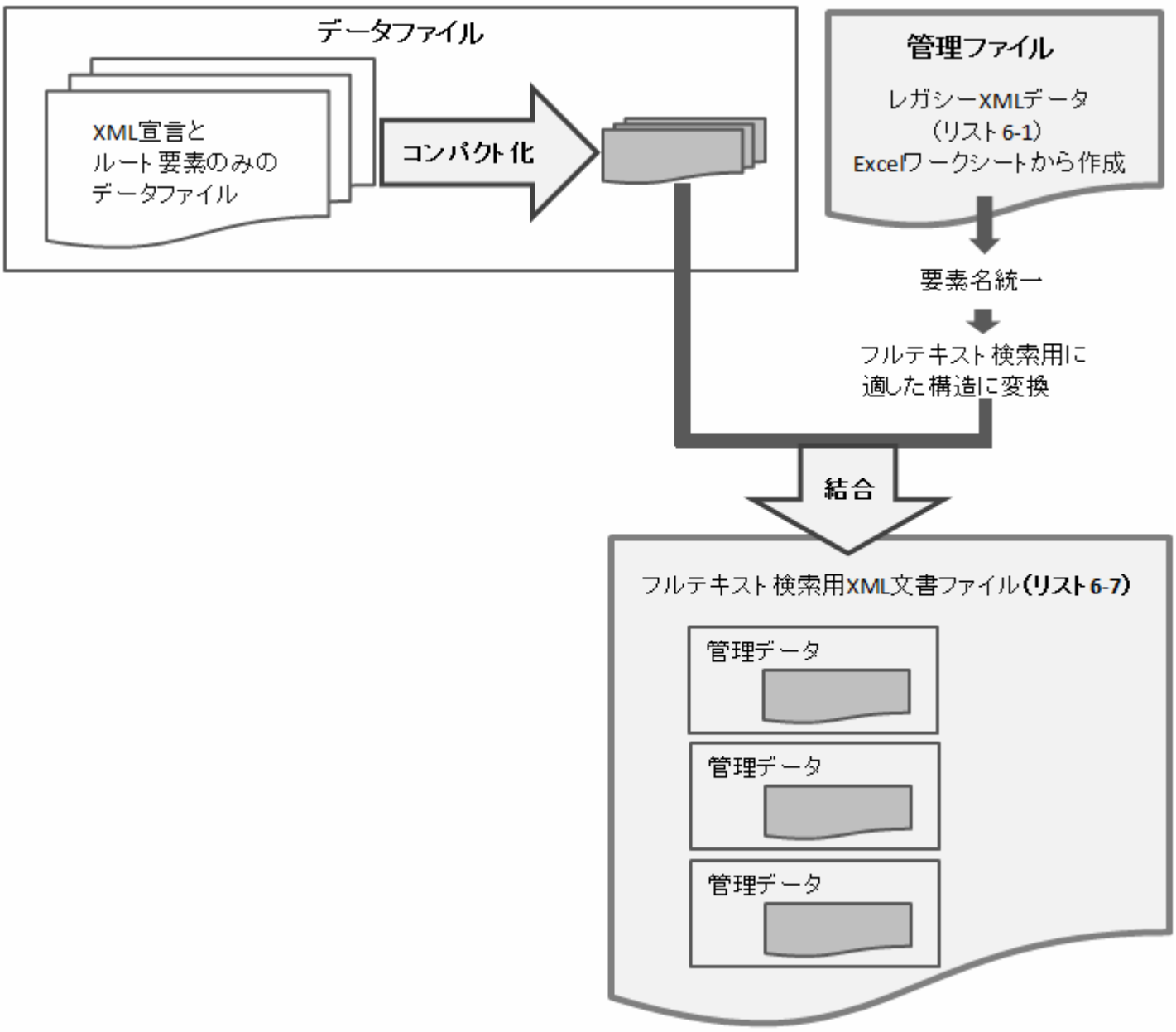
●図 6-8 絞り込み検索における XSLT の用い方



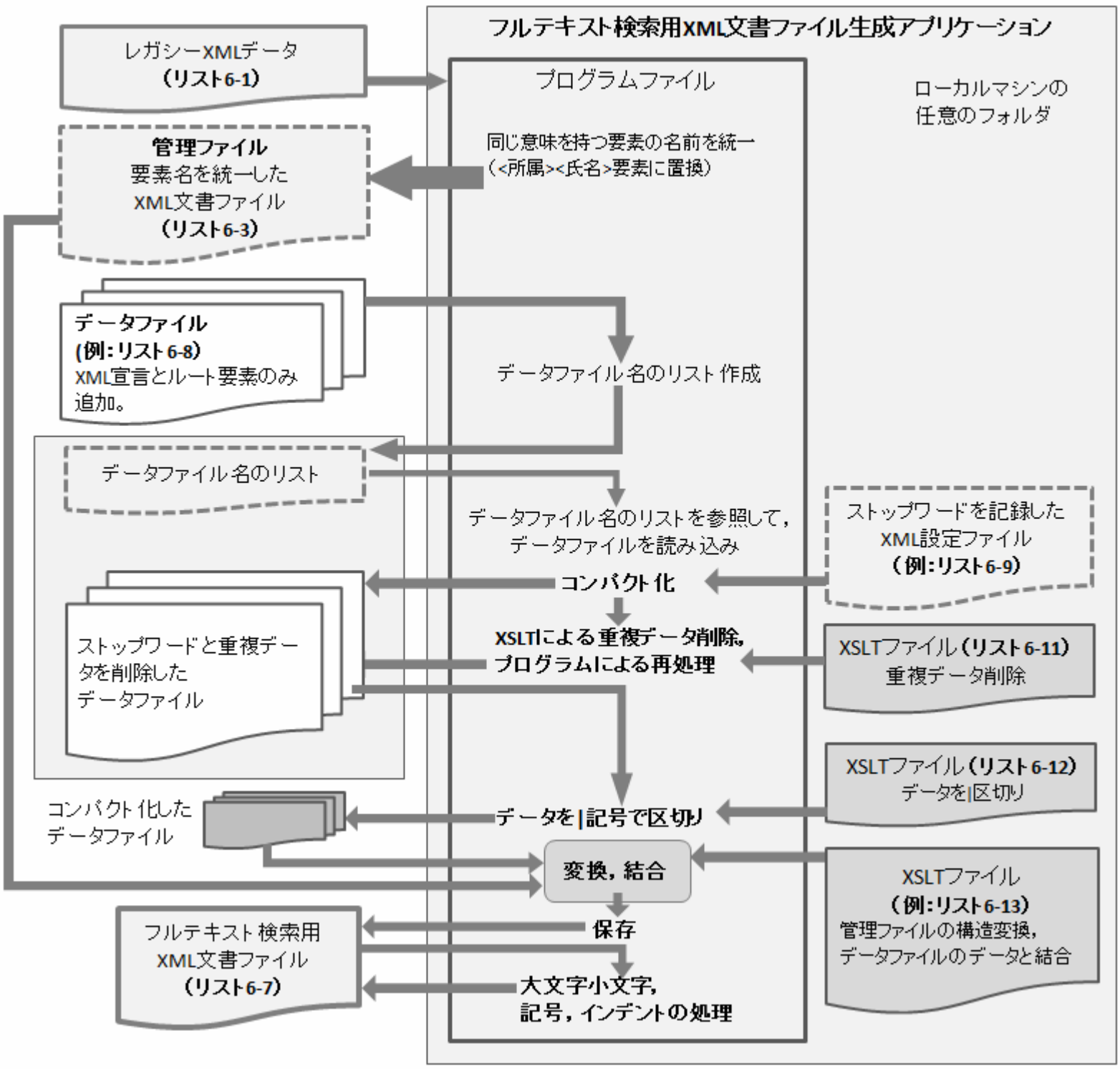
●図 6-9 本文内容の原稿テキストに，XML 宣言およびルート要素のみ追加して，データファイルを作成する



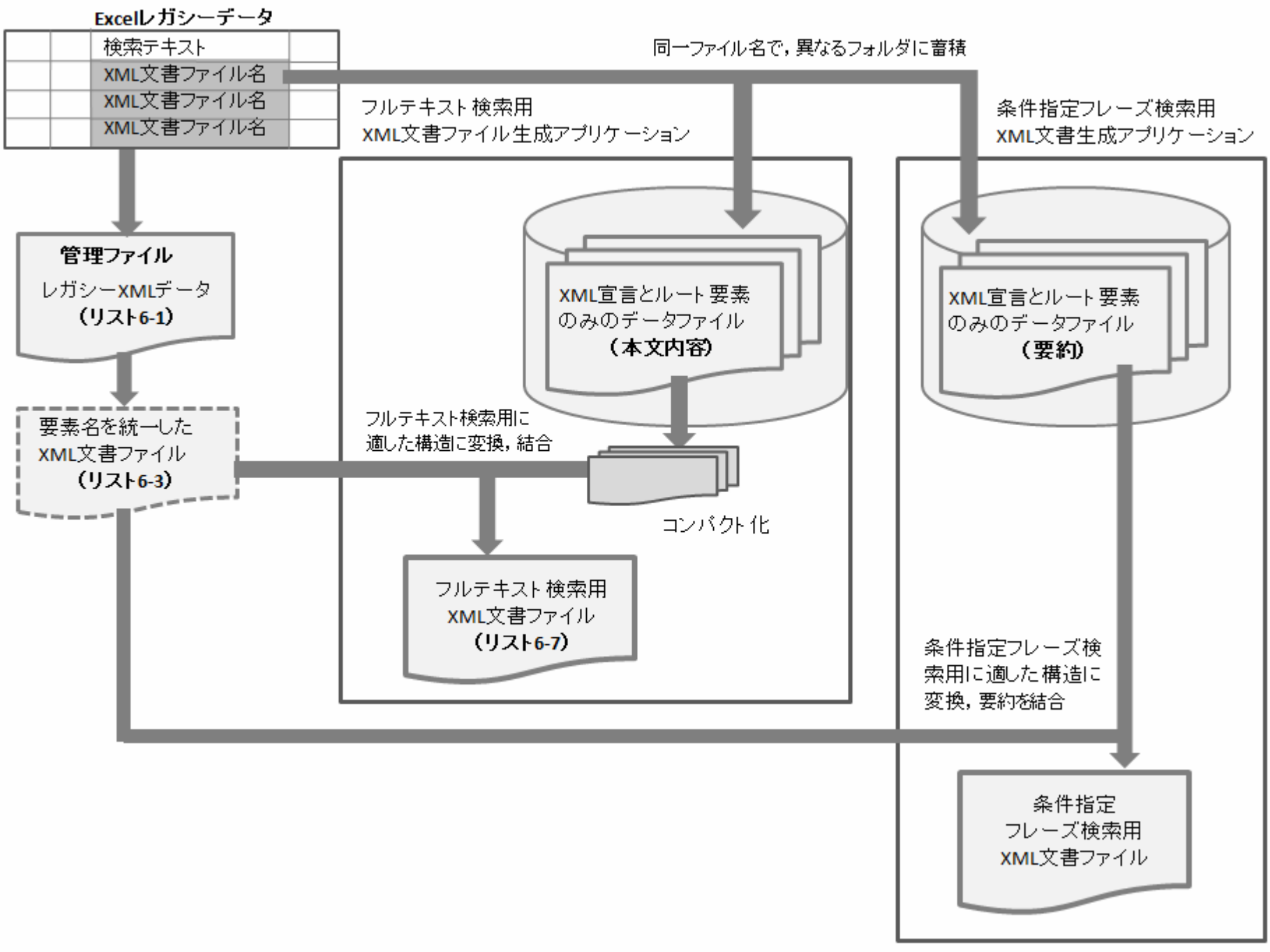
●図 6-10 検索対象とする複数の XML 文書をコンパクト化して，Excel レガシーデータと結合する



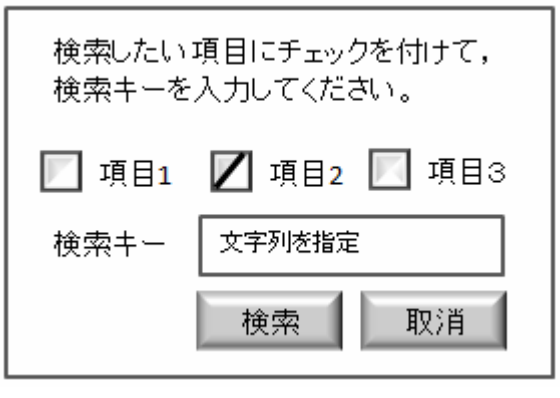
●図 6-11 ストップワード方式によるデータファイルのコンパクト化の処理の流れ



●図 6-12 Excel レガシーデータから，フルテキスト検索用と，条件指定フレーズ検索用のファイルを別個に作成する。



●図 6-13 ユーザがデータの有無を条件として指定する，条件指定検索の画面の例



●図 6-14 CheckBox の選択パターンと対応させる疑似インデックス

	CheckBox1	CheckBox2	CheckBox3	属性
	要素1	要素2	要素3	
1	◪	◪	◪	インデックス="111"
2	◪	◪	□	インデックス="110"
3	◪	□	◪	インデックス="101"
4	◪	□	□	インデックス="100"
5	□	◪	◪	インデックス="011"
6	□	◪	□	インデックス="010"
7	□	□	◪	インデックス="001"
8	□	□	□	インデックス="000"

●図 6-15 データ後の空白は，空白入りのままエクスポートされる。反転部分は空白である。

B2 在宅支援

	A	B	C	D	E
1	年度	支援制度	テーマ	内容	詳細
2	平成21年度	在宅支援	プログラミング	入門	
3	平成21年度	在宅支援	プログラミング	基本	
4	平成21年度	在宅支援	プログラミング	応用	

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
  - <講座>
      <ID>1</ID>
      <年度>平成21年度</年度>
      <支援制度>在宅支援                    </支援制度>
      <テーマ>プログラミング</テーマ>
      <内容>入門</内容>
      <パス>2009</パス>
      <コンテンツ>pg_01.htm</コンテンツ>
      <検索テキスト>pg_01.xml</検索テキスト>
      <所属1>女性技術者の会</所属1>
      <氏名1>古見　久美</氏名1>
    </講座>
  - <講座>
```

●図 6-16 軸の方向とコンテキスト･ポジションに注意する

```
<?xml version="1.0"?>
<講座実施報告書>
        <目次>1月度</目次>      preceding-sibling::目次[2]
        <目次>2月度</目次>      preceding-sibling::目次[1]
        <目次>3月度</目次>
        <目次>4月度</目次>      following-sibling::目次[1]
        <目次>5月度</目次>      following-sibling::目次[2]
        <目次>6月度</目次>      following-sibling::目次[3]
</講座実施報告書>
```

●図 6-17 選択項目のインデックスに対応する，処理に必要な値をもとめる

選択項目
（例: ListBox）

6月度
5月度
4月度
3月度
2月度
1月度

選択項目の値	処理に必要な値			
プログラム	XPath		DOM	
インデックス（n）	総ノード数（a）	コンテキスト・ポジション（p）	総ノード数（a）	インデックス（i）
0	6	6	6	5
1	6	5	6	4
2	6	4	6	3
3	6	3	6	2
4	6	2	6	1
5	6	1	6	0
もとめ方	fn:count()	fn:position()	length	NodeList.item(index);

XPathを用いる場合はp=a-n，DOMを用いる場合はi=a-n-1として，処理に必要な値をもとめる。

選択された値に対応して処理するXML文書ファイル

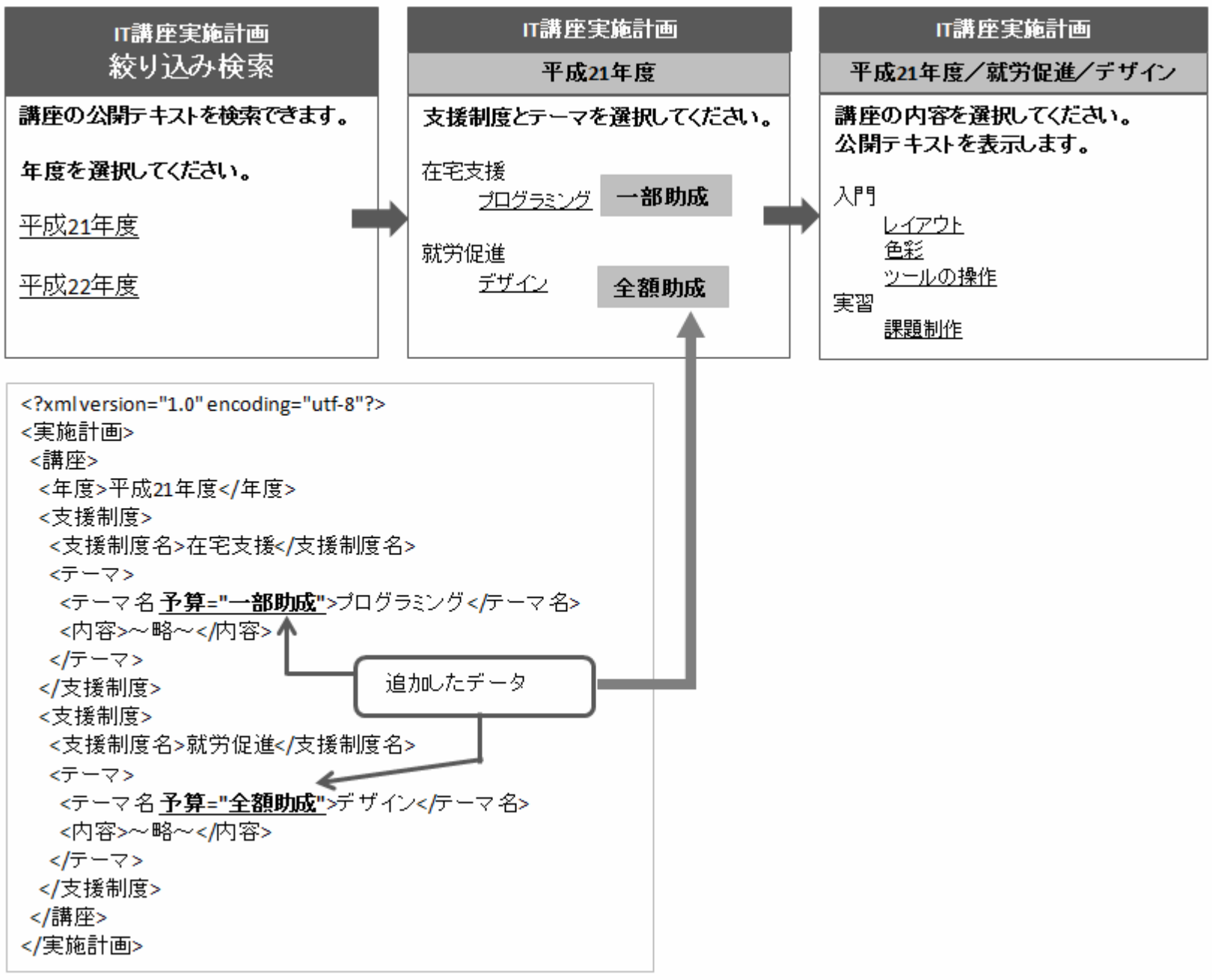
```
<?xml version="1.0"?>
<講座実施報告書>
        <目次>1月度</目次>
        <目次>2月度</目次>
        <目次>3月度</目次>
        <目次>4月度</目次>
        <目次>5月度</目次>
        <目次>6月度</目次>
</講座実施報告書>
```

●図 6-18 XML ソース中のコメントのネストはできない。

●図 6-19 運用開始後に「助成方法」フィールドが中途階層に追加されている。

年度	支援制度	テーマ	助成方法	内容	詳細
平成21年度	在宅支援	プログラミング	一部助成	入門	
平成21年度	在宅支援	プログラミング	一部助成	基本	
平成21年度	在宅支援	プログラミング	一部助成	応用	
平成21年度	就労促進	デザイン	全額助成	入門	レイアウト
平成21年度	就労促進	デザイン	全額助成	入門	色彩
平成21年度	就労促進	デザイン	全額助成	入門	ツールの操作
平成21年度	就労促進	デザイン	全額助成	実習	課題制作
平成22年度	在宅支援	データベース	一部助成	概論	
平成22年度	在宅支援	データベース	一部助成	実践	論理設計
平成22年度	在宅支援	データベース	一部助成	実践	物理設計

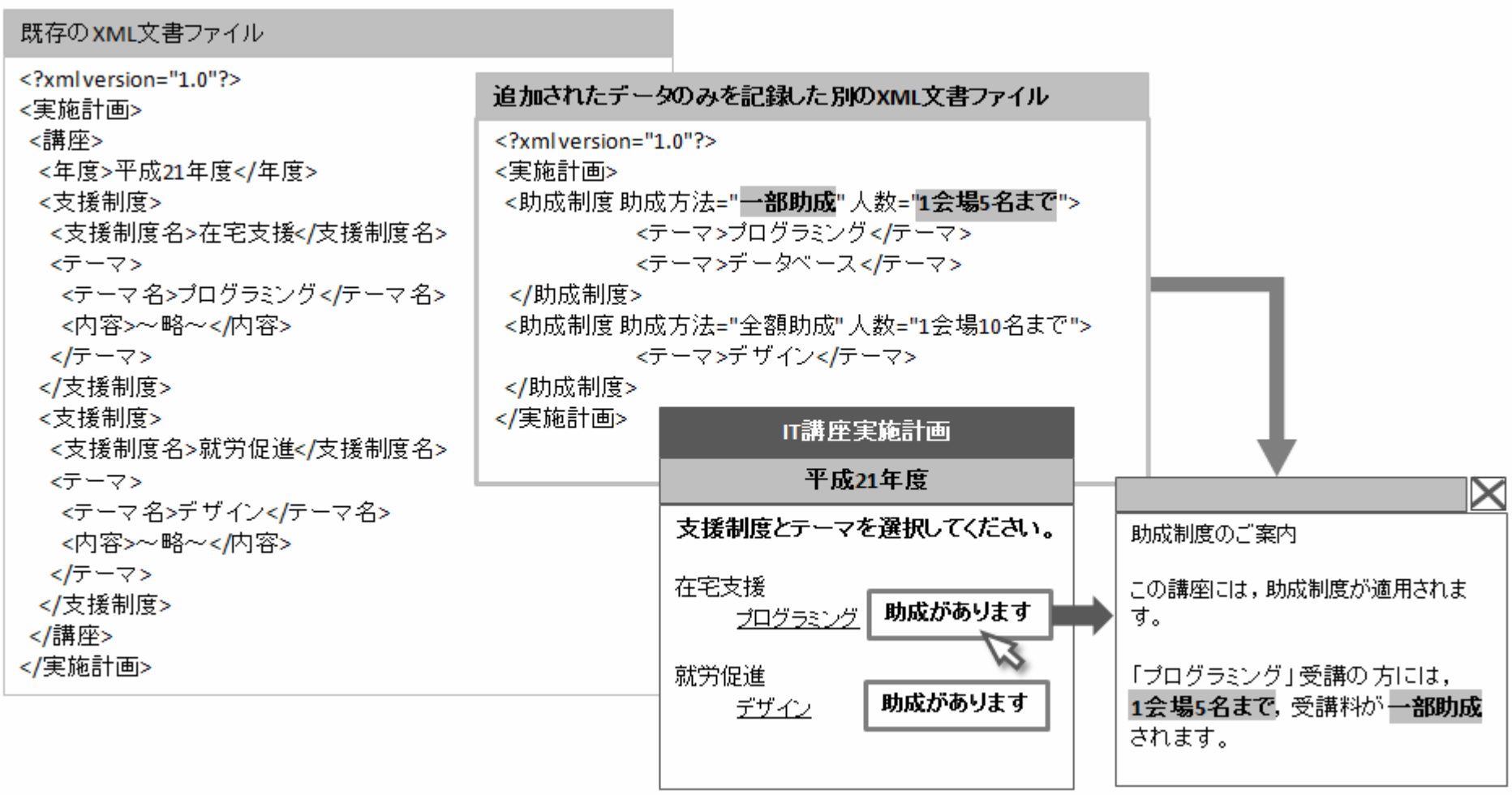
●図 6-20 属性値として追加し，表示処理の追加で対応する方法。



●図 6-21 運用開始後に「助成方法」「支援枠」フィールドが中途階層に追加されている。

年度	支援制度	テーマ	助成方法	支援枠	内容
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	入門
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	基本
平成21年度	在宅支援	プログラミング	一部助成	1会場5名まで	応用
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	入門
平成21年度	就労促進	デザイン	全額助成	1会場10名まで	実習
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	概論
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	実践
平成22年度	在宅支援	データベース	一部助成	1会場5名まで	実践

●図 6-22 別の XML 文書ファイルを作成して，追加の処理で対応する方法。



●図 6-23 既存項目の文字列を含む新規項目が追加された例。

	A	B	C	D	E	F
1	年度	支援制度	テーマ	内容	詳細	パス
2	平成21年度	在宅支援	プログラミング	入門		2009
3	平成21年度	在宅支援	プログラミング	基本		2009
4	平成21年度	在宅支援	プログラミング	応用		2009
5	平成21年度	就労促進	**デザイン**	入門	レイアウト	2009
6	平成21年度	就労促進	**デザイン**	入門	色彩	2009
7	平成21年度	就労促進	**デザイン**	入門	ツールの操作	2009
8	平成21年度	就労促進	**デザイン**	実習	課題制作	2009
9	平成21年度	就労促進	インタラクティブ**デザイン**	入門		2009
10	平成21年度	就労促進	インタラクティブ**デザイン**	基本		2009
11	平成22年度	在宅支援	データベース	概論		2010
12	平成22年度	在宅支援	データベース	実践	論理設計	2010
13	平成22年度	在宅支援	データベース	実践	物理設計	2010