[[strawman: catch_guards]]

Rationale

Table of Contents

- Rationale
- Syntax
- Restrictions
- Translation
  - Without unconditional catch, without finally
  - Without unconditional catch, with finally
  - With unconditional catch, without finally
  - With unconditional catch, with finally

Programs often need to catch exceptions conditionally. In portable ES, they have to write it like so:

```
try {
    ...
} catch (e) {
    if (p(e)) {
        ...
    } else {
        throw e;
    }
}
```

SpiderMonkey supports a simple syntax for *catch guards*, which abstract over this pattern. With catch guards the above could be rewritten:

```
try {
    ...
} catch (e if p(e)) {
    ...
}
```

This is simpler, more readable, and less error-prone (no forgetting to rethrow). It also means that if implementations choose to attach meta-data at `throw` points, there's no danger in overwriting this metadata by explicit rethrows.

Syntax

The syntax of `try-catch[-finally]` statements changes to:

```
TryStatement ::= "try" Block FinallyClause
               |  "try" Block ConditionalCatchClause+ UnconditionalCatchClause?
FinallyClause?
               |  "try" Block UnconditionalCatchClause FinallyClause?

UnconditionalCatchClause ::= "catch" "(" Identifier ")" Block
ConditionalCatchClause   ::= "catch" "(" Identifier "if" Expression ")" Block
FinallyClause            ::= "finally" Block
```

Restrictions

Note that the above syntax precludes:

- zero catch clauses without a finally clause
- catch clauses following unconditional catches

Translation

The translations in this section assume an invisible identifier `$tmp` that is not visible even to direct `eval`. They use the notation `Expression[x -> y]` to mean scope-respecting substitution of `y` for `x` in the expression.

Without unconditional catch, without finally

A statement of the form:

```
try B0
catch (x1 if e1) B1
...
catch (xn if en) Bn
```

translates to:

```
try B0
catch ($tmp) {
    if (e1[x1 --> $tmp]) B1
    else if ...
    else if (en[xn --> $tmp]) Bn
    else throw $tmp;
}
```

Without unconditional catch, with finally

A statement of the form:

```
try B0
catch (x1 if e1) B1
...
catch (xn if en) Bn
finally BB
```

translates to:

```
try B0
catch ($tmp) {
    if (e1[x1 --> $tmp]) B1
```

```
    else if ...
    else if (en[xn --> $tmp]) Bn
    else throw $tmp;
} finally BB
```

With unconditional catch, without finally

A statement of the form:

```
try B0
catch (x1 if e1) B1
...
catch (xn if en) Bn
catch BB
```

translates to:

```
try B0
catch ($tmp) {
    if (e1[x1 --> $tmp]) B1
    else if ...
    else if (en[xn --> $tmp]) Bn
    else BB
}
```

With unconditional catch, with finally

A statement of the form:

```
try B0
catch (x1 if e1) B1
...
catch (xn if en) Bn
catch BB
finally BBB
```

translates to:

```
try B0
catch ($tmp) {
    if (e1[x1 --> $tmp]) B1
    else if ...
    else if (en[xn --> $tmp]) Bn
    else BB
} finally BBB
```