RSVP.PROMISE CLASS

DEFINED IN: bower_components/rsvp/lib/rsvp/promise.js:30
MODULE: ember

Promise objects represent the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its `then` method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled.

TERMINOLOGY

- `promise` is an object or function with a `then` method whose behavior conforms to this specification.
- `thenable` is an object or function that defines a `then` method.
- `value` is any legal JavaScript value (including undefined, a thenable, or a promise).
- `exception` is a value that is thrown using the throw statement.
- `reason` is a value that indicates why a promise was rejected.
- `settled` the final resting state of a promise, fulfilled or rejected.

A promise can be in one of three states: pending, fulfilled, or rejected.

Promises that are fulfilled have a fulfillment value and are in the fulfilled state. Promises that are rejected have a rejection reason and are in the rejected state. A fulfillment value is never a thenable.

Promises can also be said to *resolve* a value. If this value is also a promise, then the original promise's settled state will match the value's settled state. So a promise that *resolves* a promise that rejects will itself reject, and a promise that *resolves* a promise that fulfills will itself fulfill.

BASIC USAGE:

```
var promise = new Promise(function(resolve, reject) {
  // on success
  resolve(value);

  // on failure
  reject(reason);
});

promise.then(function(value) {
  // on fulfillment
}, function(reason) {
  // on rejection
});
```

ADVANCED USAGE:

Promises shine when abstracting away asynchronous interactions such as `XMLHttpRequest`s.

```
function getJSON(url) {
  return new Promise(function(resolve, reject){
    var xhr = new XMLHttpRequest();

    xhr.open('GET', url);
    xhr.onreadystatechange = handler;
    xhr.responseType = 'json';
    xhr.setRequestHeader('Accept', 'application/json');
    xhr.send();

    function handler() {
      if (this.readyState === this.DONE) {
        if (this.status === 200) {
          resolve(this.response);
        } else {
          reject(new Error('getJSON: `' + url + '` failed with status: [' + this.status + ']'));
        }
      }
    };
  });
}

getJSON('/posts.json').then(function(json) {
  // on fulfillment
}, function(reason) {
  // on rejection
});
```

Unlike callbacks, promises are great composable primitives.

```
Promise.all([
  getJSON('/posts'),
  getJSON('/comments')
]).then(function(values){
  values[0] // => postsJSON
  values[1] // => commentsJSON

  return values;
});
```

Index | Methods

Show: ✓ Inherited ☐ Protected ☐ Deprecated

METHODS

catch
finally
then