Zone Proposal

for TC39 January 2016

(spec proposal)

What are we solving?

- Writing async code is hard.
- Hard to ensure that when async code runs proper bookkeeping is done (framework rendering; action tracking; performance; testing)
- We can make writing async applications easier.
- Coexistence of different code on the same environment.

What is a Zone?

Synonyms: Zone, Domain, Context, “async local storage” (~ thread LS)

Definition: "Context" propagation through logically connected asynchronous operations

Existing Implementations

- Zone.js (Angular), Zone (StrongLoop)
- Dart Zones
- Node.js:
  - domains
  - continuation local storage
  - asyncwrap
- C# logical call context
- (Kinda) Go goroutines

https://docs.google.com/document/d/1tlQ0R6wQFGqCS5KeIw0ddoLbaSYx6aU7vyXOkv-wvlM/edit#

Motivations

- Frameworks:
  - Automatically tracking rendering at the end of async operations.
- Testing:
  - Knowing when to proceed with assertions of tests (by tracking number of outstanding tasks)
- Application User Tracking/Timing:
  - Knowing how much user time, user actions take.
- Async-Local Error Handling:
  - Consistent way of capturing of errors on per zone basis.

"Context" Propagation

```
Zone.current; // Globally-accessible property for the current context.

let z = Zone.current;

Promise.resolve(0).then(() => {
  assert(Zone.current === z);
});
```

"Context" Propagation

```
let z1 = Zone.current;
```

```
let z2 = Zone.current;
```

```
assert(z1 !== z2);
```

```
Promise.resolve(0).then(() => {
  assert(Zone.current === z1);
});
```

```
Promise.resolve(0).then(() => {
  assert(Zone.current === z2);
});
```

API

```
class Zone {
  // current zone instance
  static get current();

  // Wraps a given function, returning a new function which
  // restores the current zone before calling the original
  wrap(callback);
}
```

"Context" Propagation

```
Zone.current; // Global variable pointing to the current context.

let z = Zone.current;

Promise.resolve(0).then(Zone.current.wrap(() => {
  expect(Zone.current).toBe(z);
}));
```

Naive `Promise.prototype.then` changes

```
const originalThen = Promise.prototype.then;

Promise.prototype.then = function (onFulfilled, onRejected) {
  onFulfilled = Zone.current.wrap(onFulfilled);
  onRejected = Zone.current.wrap(onRejected);

  return originalThen.call(this, onFulfilled, onRejected);
};
```

Now it just works

```
let z = Zone.current;

Promise.resolve(0).then(() => {
  expect(Zone.current).toBe(z);
});
```

- Similar thing needs to be done with web APIs.
  - (And libraries that do custom scheduling also need to do this.)

More Realistic Example

```
Zone.current.fork({name: 'data-request'}, {
  rURL: url,
  rTime: new Date().getTime()
}).run(() => {
  fetch(url).
    then(renderFn).
    then(() => {
      var z = Zone.current;
      var now = new Date().getTime();
      console.log(z.get('rUrl'), 'took', z.get('rTime') - now);
    });
});
```

MVP API

```
class Zone {
  static get current();

  wrap(callback); // create a version of callback that enters/runs/exits
  run(callback); // enter a zone, run the callback, exit the zone
  fork(zoneSpec, properties); // for creating new (nested) zones
  get(key);
}
```

Changes needed to the language

- Promise.prototype.then
  - Save current zone along with the handlers; use zone.run to invoke them later

- async / await
  - Ensure the behind-the-scenes promise handlers automatically save and restore `Zone.current`.

Why can't this be left to libraries?

- `async/await` and promises
- Needs to be in place for platform environments to take advantage of it
- It only works if all third-party libraries with custom scheduling call `Zone.current.wrap(...)`.
- We need to get library owners to think in terms of async context propagation.

Links

- Zone Proposal: https://gist.github.com/mhevery/63fdcdf7c65886051d55

Q & A

<support slides>

Forking

```
class Zone {
  // Creates and returns a child Zone with additional behavior.
  // zoneSpec is an object specifying these set of behaviors.
  fork(zoneSpec);

  wrap(callback);
  static get current();
}
```

Running a callback in a zone

```
let savedZone = Zone.current;
Zone.current = newZone;
try {
  doWork(...args);
} finally {
  Zone.current = oldZone;
}
```

- Error prone / hard to get it right
- Doesn’t work unless we add a setter for `Zone.current`.

Entering Zone

```
class Zone {
  // Execute a function in this zone. Returns its return value
  run(callback);

  fork(zoneSpec);
  wrap(callback);
  static get current();
}
```

Property retrieval

```
class Zone {
  // Retrieves the value for a key, walking parent zones.
  get(key);

  run(callback);
  fork(zoneSpec);
  wrap(callback);
  static get current();
}
```

Value Proposition

- Tracking async local variables through async operations.

What if in addition to just storing data on zone, we would allow interception and error handling?

Callback interception

- If we can intercept the callback wrapping, then we could run code before and after callback.

- Value Proposition:
  - Zone local exception handling
  - Zone local before/after hooks (framework render hooks)
  - Zone local debugging (logging, long stack traces, latency reporting)

Error Handling

```
class Zone {
  /// @param [error] Caught exception.
  /// @returns true if error should be rethrown.
  handleError(error);

  run(callback);
  fork(zoneSpec);
  wrap(callback);
  get(key);
  static get current();
}
```

Callback Interception

Choices:

1. Don’t allow callback interception (MVP)

2. Allow users to monkey patch `Zone.current.wrap()` by reading and invoking that property from the appropriate place

3. Spec out proper APIs
   Safer / Composable / Libraries can cooperate

Forking

```
Zone.current.fork({
  name,
  onIntercept(...) { … },
  onInvoke(...) { … },
  onHandleError(...) { … },
  onFork(...) { … }
});
```

Running a callback in a zone w/ interception

```
run(delegate, applyThis, applyArgs) {
  let savedZone = Zone.current;
  Zone.current = newZone;
  try {
    newZone.[[zoneSpec]].onInvoke(delegate, applyThis, applyArgs);
  } catch (e) {
    newZone.[[zoneSpec]].onHandleError(e);
  } finally {
    Zone.current = oldZone;
  }
}
```

API Proposal

```
class Zone {
  get static current();

  fork(zoneSpec);
  wrap(callback);
  run(callback);
  handleError(error);
  get(key);
}
```