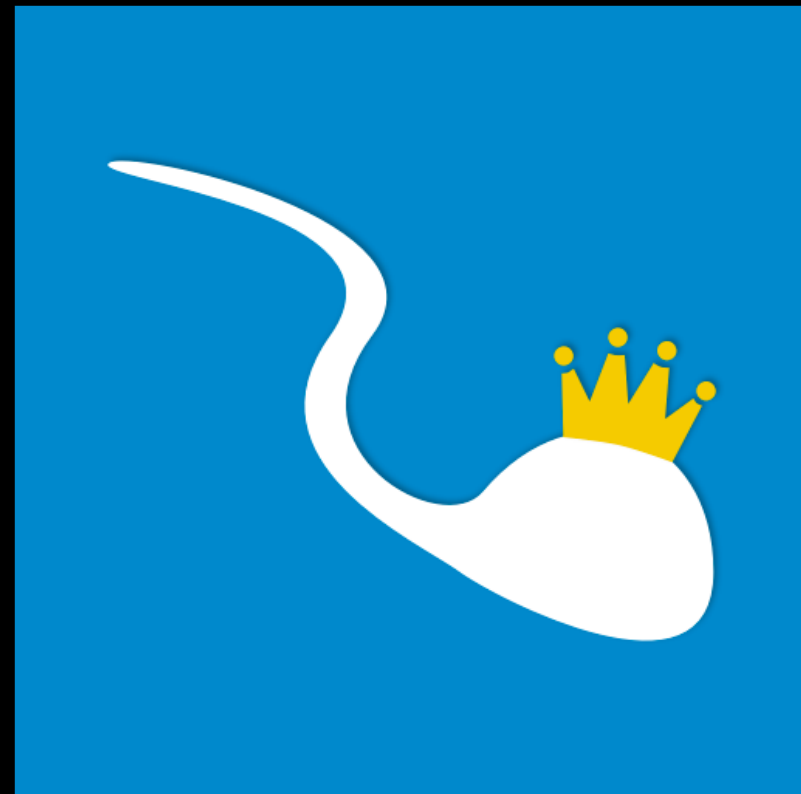


# Node.js 错误处理实践

—— 把「异常」当作「日常」

# 关于我

- 王子亭
- 20 岁
- Node.js 开发者
- LeanCloud
- <https://jysperm.me>
- GitHub: jysperm



# 目标

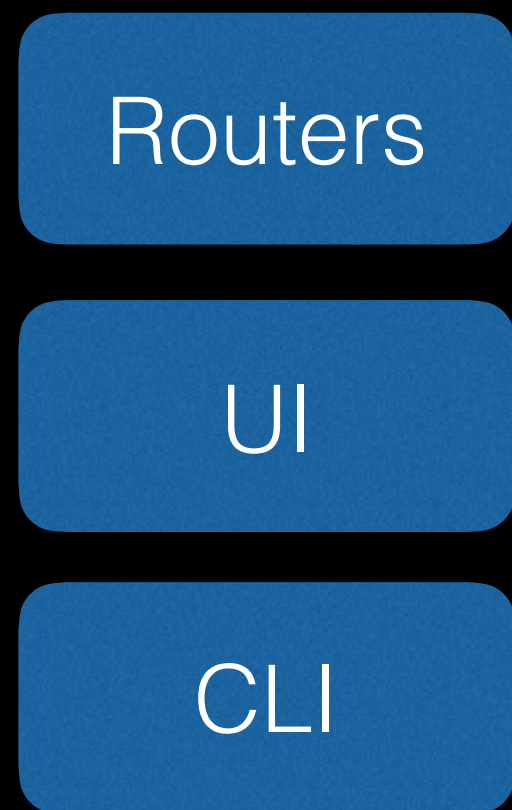
- 出现错误时能将任务中断在一个合适的位置
- 能记录错误的摘要、调用栈以及其他上下文
- 通过这些记录能够快速地发现和解决问题

# 而不是

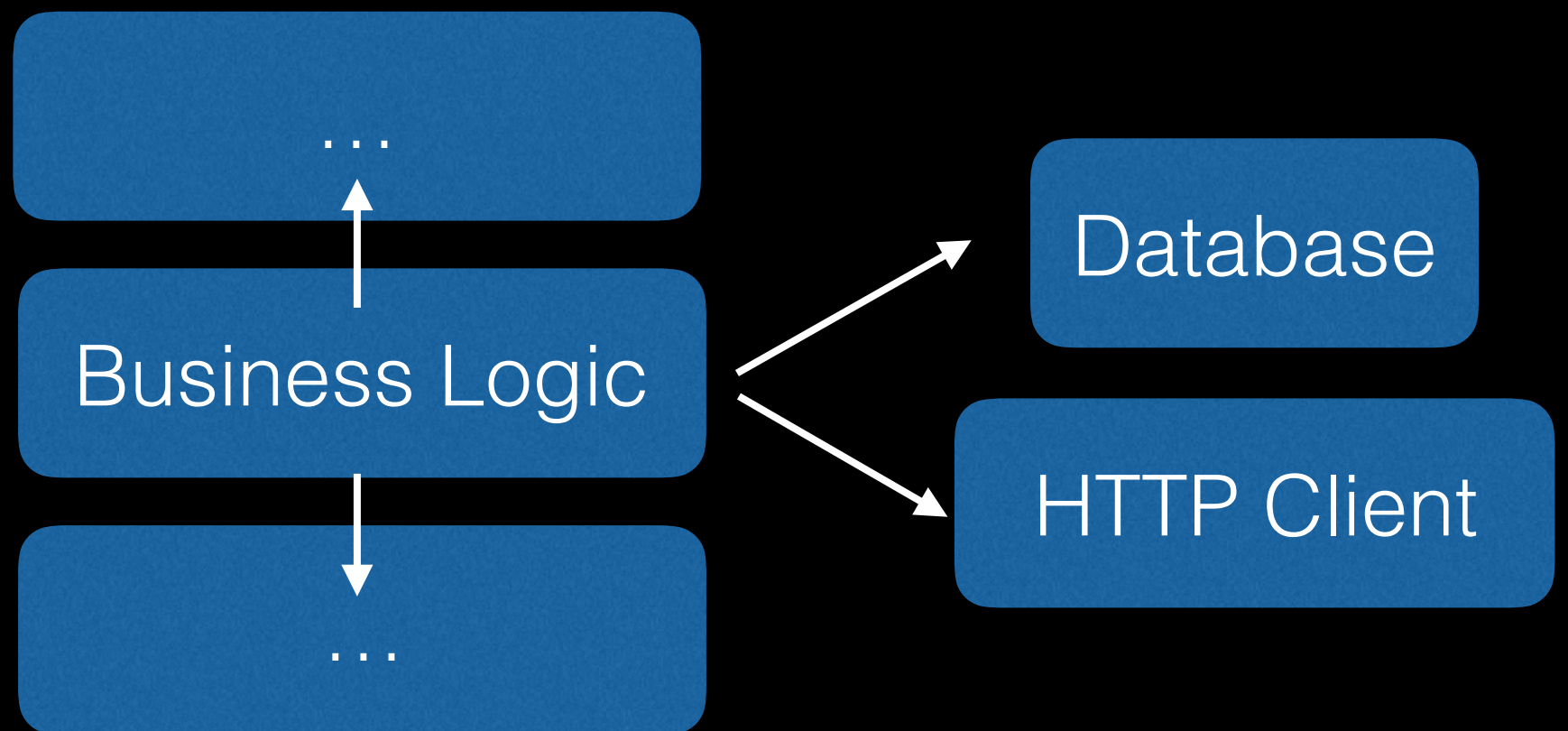
- 出现错误后程序崩溃退出
- 出现错误后 HTTP 请求无响应
- 出现错误后数据被修改了「一半」，出现不一致
- 出现错误后没有记录日志或重复记录
- 在日志中打印了错误但没有提供调用栈和上下文

# 层次化架构

Dispatcher



Data Access



# Exception

```
try {  
  step1();  
} catch (err) {  
  console.error(err.stack);  
}  
  
function step1() {  
  // ...  
  step2();  
  // ...  
}  
  
function step2() {  
  if ( ... )  
    throw new Error('some error');  
}
```

```
var err = step1();  
if (err) console.error(err);  
  
function step1() {  
  // ...  
  var err = step2();  
  if (err) return 'step1: ' + err;  
  // ...  
}  
  
function step2() {  
  if ( ... )  
    return 'step2: some error';  
}
```

# Stack Trace

```
Error: some error  
  at step2  (~/exception.js:14:9)  
  at step1  (~/exception.js:9:3)  
  at <anonymous> (~/exception.js:2:3)
```

```
step1: step2: some error
```

- 预期的异常：参数不合法、前提条件不满足；通常主动 throw。
- 非预期的异常：JavaScript 引擎的运行时异常、来自依赖库的异常。



# throw Exception

- 总是 throw 一个继承自 Error 的对象
- 慎用自定义的异常类型\*
- 可以直接向异常上附加属性来提供上下文

- 可以直接向异常上附加属性来提供上下文

```
var err = new Error('Permission denied');  
err.statusCode = 403;  
throw err;
```

```
var err = new Error('Error while downloading');  
err.url = url;  
err.responseCode = res.statusCode;  
throw err;
```

# Asynchronous

- Node.js style callback
- Promise (co/generator、async/await)
- EventEmitter (Stream)

Expected

```
root {  
  ·· copyFileContent() {  
    ·· ·· fs.readFile() {  
    ·· ·· ·· fs.writeFile()  
    ·· ·· }  
  ·· }  
}
```

Actual

```
root {  
  ·· copyFileContent()  
  ·· // ...  
  ·· fs.readFile()  
  ·· // ...  
  ·· fs.writeFile()  
}
```

# Node.js style callback

```
function copyFileContent(from, to, callback) {  
  fs.readFile(from, (err, buffer) => {  
    if (err) {  
      callback(err);  
    } else {  
      try {  
        fs.writeFile(to, buffer, callback);  
      } catch (err) {  
        callback(err);  
      }  
    }  
  });  
}
```

```
try {  
  ·· copyFileContent(from, to, (err) => {  
    ·· ·· if (err) {  
      ·· ·· ·· console.error(err);  
    ·· ·· } else {  
      ·· ·· ·· console.log('success')  
    ·· ·· }  
    ·· }  
  ·· });  
} catch (err) {  
  ·· console.error(err);  
}
```

# 琐碎的细节

- 需要同时处理同步异常和异步回调
- 在每次回调中检查 `err` 的值（有值提前 `return`）
- 回调中的代码也需要捕捉同步异常
- 确保无论成功或失败，要么 `callback` 被调用，要么同步地抛出异常

# Promise

```
function copyFileContent(from, to) {  
  return fs.readFile(from).then( (buffer) => {  
    return fs.writeFile(to, buffer);  
  });  
}
```

```
Promise.try( () => {  
  return copyFileContent(from, to);  
}).then( () => {  
  console.log('success');  
}).catch( (err) => {  
  console.error(err);  
});
```



- 避免手动创建 Promise

```
function copyFileContent(from, to) {  
  return new Promise( (resolve, reject) => {  
    fs.readFile(from, (err, buffer) => {  
      if (err) {  
        reject(err);  
      } else {  
        try {  
          fs.writeFile(to, buffer, resolve);  
        } catch (err) {  
          reject(err);  
        }  
      }  
    });  
  });  
}
```

- 尽量使用 Promise 库提供的工具函数去调用 callback 风格代码

```
function copyFileContent(from, to) {  
  return Promise.promisify(fs.readFile)(from).then( (buffer) => {  
    return Promise.promisify(fs.writeFile)(to, buffer);  
  });  
}
```

# co/generator

```
var copyFileContent = co.wrap(function*(from, to) {  
  return yield fs.writeFile(to, yield fs.readFile(from));  
});  
  
co(function*() {  
  try {  
    console.log(yield copyFileContent(from, to));  
  } catch (err) {  
    console.error(err);  
  }  
});
```

# async/await

```
async function copyFileContent(from, to) {  
  · return await fs.writeFile(to, await fs.readFile(from));  
}  
  
try {  
  · console.log(await copyFileContent(from, to));  
} catch (err) {  
  · console.error(err);  
}
```

- 异常会随着 Promise 链传递

Promise

```
Promise.try( () => {  
  · return copyFileContent(a, b);  
}).then( () => {  
  · return copyFileContent(b, c);  
}).then( () => {  
  · return copyFileContent(c, d);  
}).then( () => {  
  · console.log('success');  
}).catch( (err) => {  
  · console.error(err);  
});
```

async/await

```
try {  
  · await copyFileContent(a, b);  
  · await copyFileContent(b, c);  
  · await copyFileContent(c, d);  
  · console.log('success');  
} catch (err) {  
  · console.error(err);  
}
```

# Async Stack Trace

Without async stack trace:

```
Error: EACCES: permission denied, open 'to'  
    at Error (native)
```

Enabled async stack trace:

```
Error: EACCES: permission denied, open 'to'  
    at Error (native)
```

```
From previous event:
```

```
    at ~/test.js:15:15
```

```
    at FSReqWrap.readFileAfterClose (fs.js:380:3)
```

```
From previous event:
```

```
    at copyFileContent (~/test.js:14:28)
```

```
    at ~/test.js:20:10
```

```
13 function copyFileContent(from, to) {  
14   return fs.readFile(from).then( (buffer) => {  
15     return fs.writeFile(to, buffer);  
16   });  
17 }  
18  
19 Promise.try( () => {  
20   return copyFileContent('a', 'b');  
21 }).then( () => {  
22   console.log('success');  
23 }).catch( (err) => {  
24   console.error(err.stack);  
25 });
```

Error: EACCES: permission denied, open 'to'  
at Error (native)

From previous event:

at ~/test.js:15:15

at FSReqWrap.readFileAfterClose (fs.js:380:3)

From previous event:

at copyFileContent (~/test.js:14:28)

at ~/test.js:20:10

# EventEmitter

```
var redisClient = redis.createClient();  
  
redisClient.on('error', (err) => {  
  console.error(err);  
});
```



# Stream

```
try {  
  · var source = fs.createReadStream(from);  
  · var target = fs.createWriteStream(to);  
  
  · source.on('error', (err) => {  
    · console.error(err);  
  · }).pipe(target).on('error', (err) => {  
    · console.error(err);  
  · });  
} catch (err) {  
  · console.log(err);  
}
```

# uncaughtException unhandledRejection

```
process.on('uncaughtException', (err) => {  
  console.error(err);  
});  
  
process.on('unhandledRejection', (reason, p) => {  
  console.error(reason, p);  
});
```

# 传递和处理异常

- 注意 Promise / callback chain 不要从中间断开
- \*只处理已知的、必须在这里处理的异常，其他异常继续向外抛出\*
- 不要轻易地丢弃一个异常
- 过程中可以向 err 对象上添加属性，补充上下文

- 只处理已知的、必须在这里处理的异常，其他异常继续向外抛出

```
function writeLogs(logs) {  
  return fs.writeFile('out/logs', logs).catch( (err) => {  
    if (err.code === 'ENOENT') {  
      return fs.mkdir('out').then( () => {  
        return fs.writeFile('out/logs', logs);  
      });  
    } else {  
      throw err;  
    }  
  });  
}
```

- 不要轻易地丢弃一个异常

```
Promise.try( () => {  
  ·· return copyFileContent('a', 'b');  
}).catch( err => {  
  ·· // ignored  
});
```

- 有时需要附加上下文之后继续抛出异常

```
function mysqlQuery(sql, placeholders) {  
  return mysqlClient.exec(sql, placeholders).catch( (err) => {  
    err.sql = sql;  
    throw err;  
  });  
}
```

- 有时需要回滚数据后继续抛出异常

```
function mysqlTransaction(transaction) {  
  return mysqlPool.getConnection( (connection) => {  
    return connection.beginTransaction().then( () => {  
      return transaction(connection).then( (result) => {  
        return connection.commit().then( () => {  
          return result;  
        });  
      }).catch( (err) => {  
        return connection.rollback().then( () => {  
          throw err;  
        });  
      });  
    });  
  });  
}
```

# 在程序的「边界」

- Routers (Web-backend)
- UI Layer (Web/Desktop App)
- Command Dispatcher (CLI Tools)



# 处理异常

- 展示错误摘要
- 发送响应、断开 HTTP 连接 (Web-backend)
- 退出程序 (CLI Tools)
- 记录日志

# Express

```
app.get('/', (req, res, next) => {  
  ·· copyFileContent(req.query.from, req.query.to).then( () => {  
  ·· | ·· res.send();  
  ·· } ).catch(next);  
});
```

```
app.use((err, req, res, next) => {  
  ·· err.userId = req.user.id;  
  ·· err.url = req.originalUrl;  
  ·· logger.error(err);  
  ·· res.status(err.statusCode || 500).send(err.message);  
});
```

# Sentry



Sentry ▾

Docs



Sentry Backend ▾

Dashboard

Stream

Releases

Settings

All Events

Bookmarks

Assigned

Sort by: Last Seen ▾

🔍 is:unresolved



24H

14D

EVENTS

USERS

Error

Error processing 'to\_python' on 'type': 'list' object has no attribute 'iteritems'



sentry.interfaces.http in format\_headers

🕒 6 minutes ago — a day old | 💬 1 | sentry



8

6

Info

Form failed to validate: {'\_\_all\_\_': [u'An error occurred while processing your credit card: Y...



stripe.payments in handle

🕒 12 minutes ago — a month old | stripe.payments



384

203

Warning

Discarded invalid value for interface: request ({u'cookies': {}, u'url': u'http://nl.labelleassie...



urllib in urlencode

🕒 2 hours ago — a month old | sentry.api



9.3k

0

Error

Error processing 'rule\_notify' on 'SlackPlugin': No connection adapters were found for 'ht...



requests.sessions in get\_adapter

🕒 28 minutes ago — a month old | 💬 1 | sentry



341

205

Error

Error processing 'rule\_notify' on 'PagerDutyPlugin': {u'status': u'invalid event', u'message'...



pygerduty in execute\_request

🕒 32 minutes ago — 6 months old | sentry



13k

0

# 小结

- 在层次化架构中，很多时候在当前层级没有足够的信息去决定如何处理错误，因此我们需要使用异常来将错误沿着调用栈逆向抛出。
- 在异步的场景下我们应该使用 Promise 或相兼容的流程控制工具来模拟异常机制。
- 传递异常时可以回滚数据或向其补充更多上下文，但如非必要，需要继续向外抛出。
- 让所有无法被恢复的错误传递到程序的「边界」处，统一处理。

# Q & A

- Node.js
- Docker
- LeanCloud
- Bitcoin
- GPG

