异常处理规范

异常是不可控的，会影响最终的呈现效果，所以要进行异常捕获

一、js语法错误，同步代码异常

对于同步运行的错误在可疑区域增加`try catch`

```
try {
    let name = 'streamax';
    console.log(nam);

} catch (error) {
    console.log(error);
}
```

但是`try catch`不能捕获语法错误，但是语法错误在开发阶段就可以看得到，编辑器也会有提示

二、ajax请求异常

1. 对于node端请求的接口数据，做默认值处理，避免当没有数据后端返回null但是前端期望是数组导致的语法错误

   ```
     @post('/fetchAbnormalFrequency')
       async fetchAbnormalFrequency() {
           const body = this.ctx.request.body;
           let data = await
   sword.service('abnormalFrequency.fetchAbnormalFrequency')({ ...body });
         // 默认值处理
           data = data || { dataList: [], dataCount: 0 };
           this.success(data);
       }
   ```

2. 与请求相关的（除了model里面的）需要加`try catch`

   ```
   try {
     axios.get();
   } catch (error) {
       console.log(error);
   }
   ```

三、静态资源加载异常

全局监控静态资源异常`window.addEventListener`

当一项资源（如图片或脚本）加载失败，加载资源的元素会触发一个 Event 接口的 error 事件，并执行该元素上的`onerror()`处理函数。这些 error 事件不会向上冒泡到 window（即不回触发`window.onerror`），不过能被单一的`window.addEventListener`捕获。

```
window.addEventListener('error', (error) => {
    console.log(error);
}, true);
<img src='/a.jpg' /> // 捕获资源加载异常
```

由于网络请求异常不会事件冒泡，因此必须在捕获阶段将其捕捉到才行

四、promise异常

在 promise 中使用 catch 可以非常方便的捕获到异步 error，没有写 catch 的 Promise 中抛出的错误无法被`onerror`或`try-catch`捕获到，所以对于每个promise都需要用catch去捕获异常

```
 axios.get().then(() => {

 }).catch(error => {
   console.log(error);
 });
```

为了防止有漏掉的 Promise 异常，建议在全局增加一个对 unhandledrejection 的监听，用来全局监听 Uncaught Promise Error

五、iframe异常

iframe的异常捕获使用`window.onerror`

```
iframe1.onerrror = (error) => {
  console.log(error);
};
```

六、React的异常捕获

React16提供了一个内置函数`componentDidCatch`，使用它可以非常简单的湖区到react下的错误信息

```
componentDidCatch(error, info) {
  console.log(error, info);
}
```

七、崩溃和卡顿

可以利用window对象的的`load`和`beforeunload`事件实现对网页崩溃的监控

```
window.addEventListener('load', function () {
      sessionStorage.setItem('good_exit', 'pending');
      setInterval(function () {
```

```
        sessionStorage.setItem('time_before_crash', new Date().toString());
    }, 1000);
});

window.addEventListener('beforeunload', function () {
    sessionStorage.setItem('good_exit', 'true');
});

if(sessionStorage.getItem('good_exit') &&
    sessionStorage.getItem('good_exit') !== 'true') {
 // 证明崩溃了
}
```

在页面加载时（`load` 事件）在 `sessionStorage` 记录 `good_exit` 状态为 `pending`，如果用户正常退出（`beforeunload` 事件）状态改为 `true`，如果崩溃 了，状态依然为 `pending`，在用户第2次访问网页的时候（第2个`load`事件），查看 `good_exit` 的状态，如果仍然是 `pending` 就是可以断定上次访问网页崩溃了

终极解决方案： 基于service worker的崩溃统计方案 https://juejin.im/entry/5be158116fb9a049c6434f4a