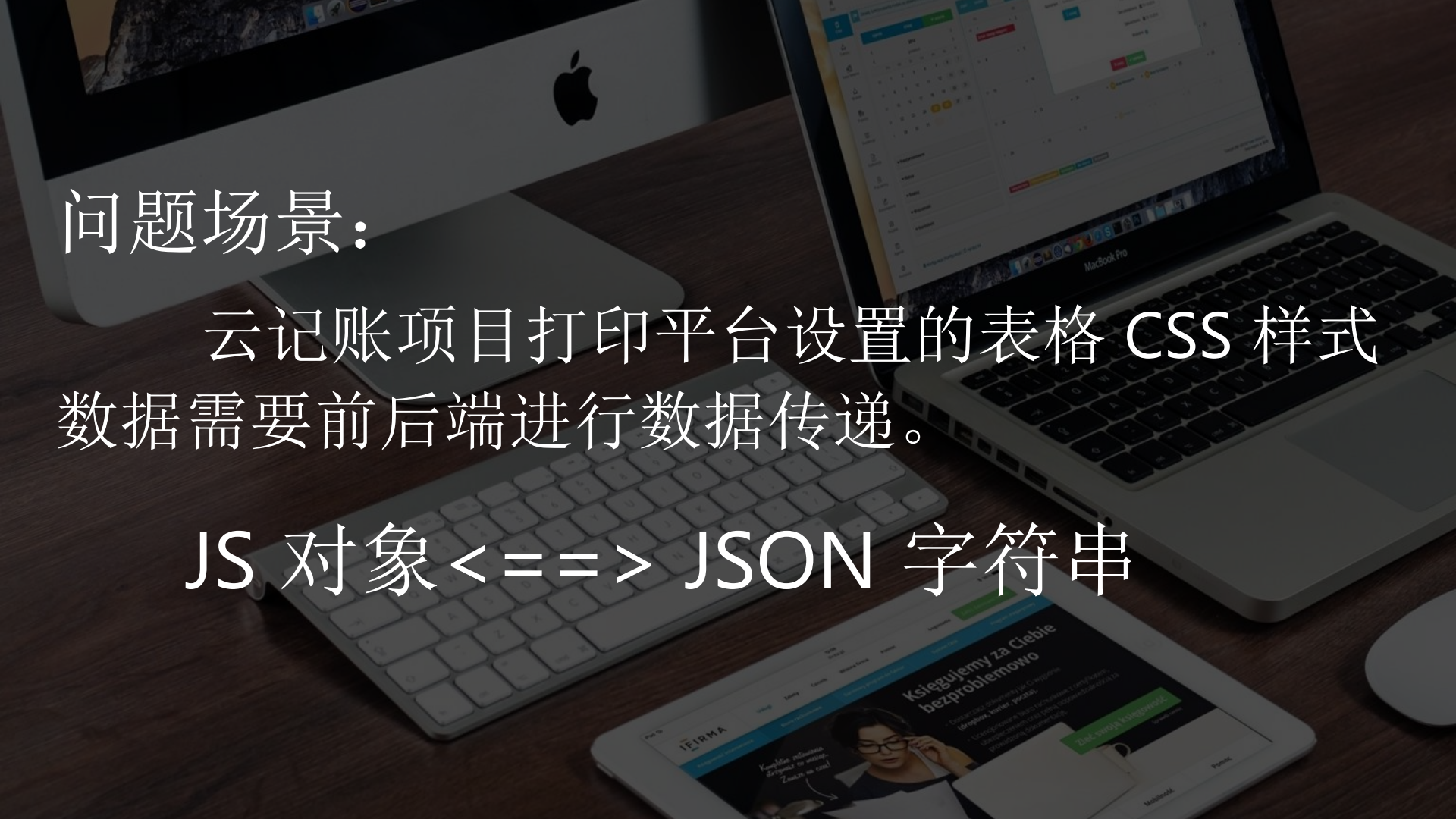
CSS的抽象语法树应用



问题场景：

云记账项目打印平台设置的表格 CSS 样式数据需要前后端进行数据传递。

JS 对象<==> JSON 字符串

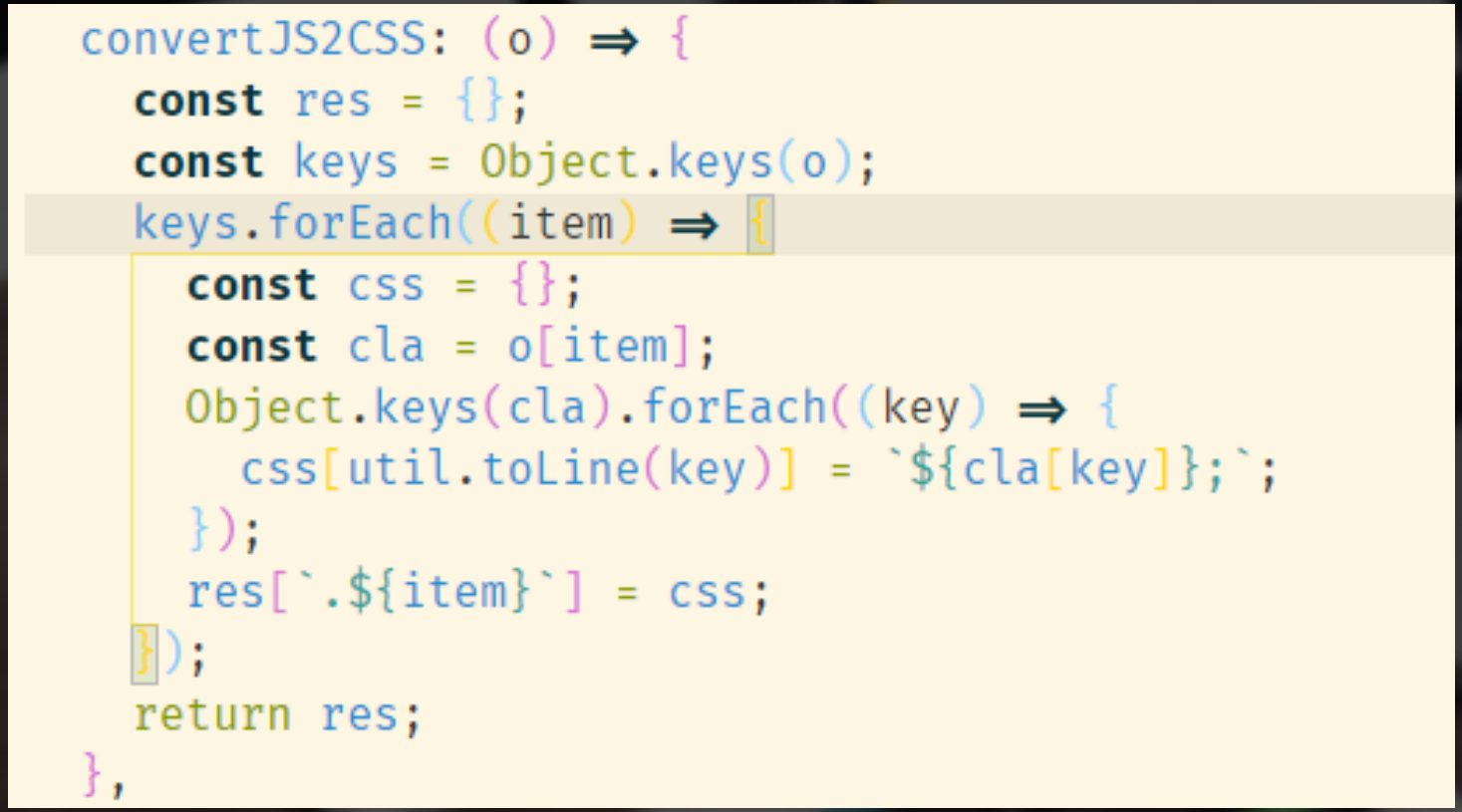
前端在设置表格样式信息时使用了一个JS对象来保存样式信息，如下图所示：

```
const initTableStyle = {
  ...thStyle,
  table: {
    fontFamily: "'SimSun', 'Courier New', Courier, monospace",
    borderCollapse: 'collapse',
    borderSpacing: '0px',
  },
  td: {
    border: '1px solid #000',
    padding: 0,
  },
  ...templateDetailsStyle,
};
```

当传递样式数据给后端时，对象中的数据需要进行一些转换，转换为 CSS 字符串。 这块就是简单的字符串替换处理：如驼峰转换下划线、：{ 等一些符号的处理。

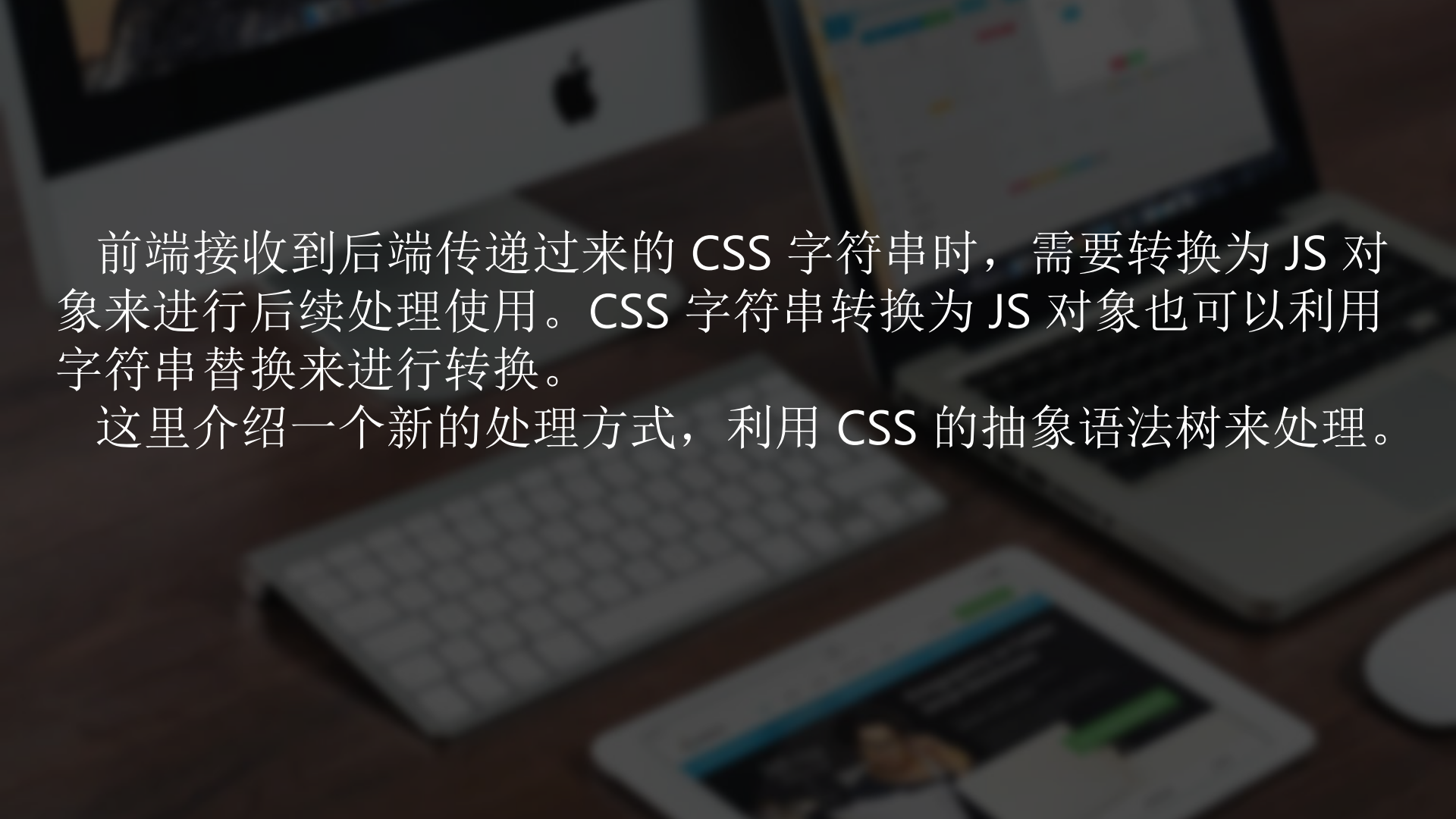
```
// 驼峰转换下划线
toLine: (name) => {
  return name.replace(/([A-Z])/g, '-$1').toLowerCase();
},
```

```
convert: (obj) => {
  const output = util.convertJS2CSS(obj);
  let value = JSON.stringify(output, null, 2)
    .replace(/",/g, '')
    .replace(/"/g, '')
    .replace(/: {/g, ' {')
    .replace(/},/g, '}')
    .replace(/rem/g, '');
  // 去掉字符串前后大括号
  value = value.substring(1);
  value = value.substring(0, value.length - 1);
  return value;
},
```


convertJS2CSS: (o) ⇒ {
const res = {};
const keys = Object.keys(o);
keys.forEach((item) ⇒ {
const css = {};
const cla = o[item];
Object.keys(cla).forEach((key) ⇒ {
css[util.toLine(key)] = `${cla[key]};`;
});
res[`.${item}`] = css;
});
return res;
},

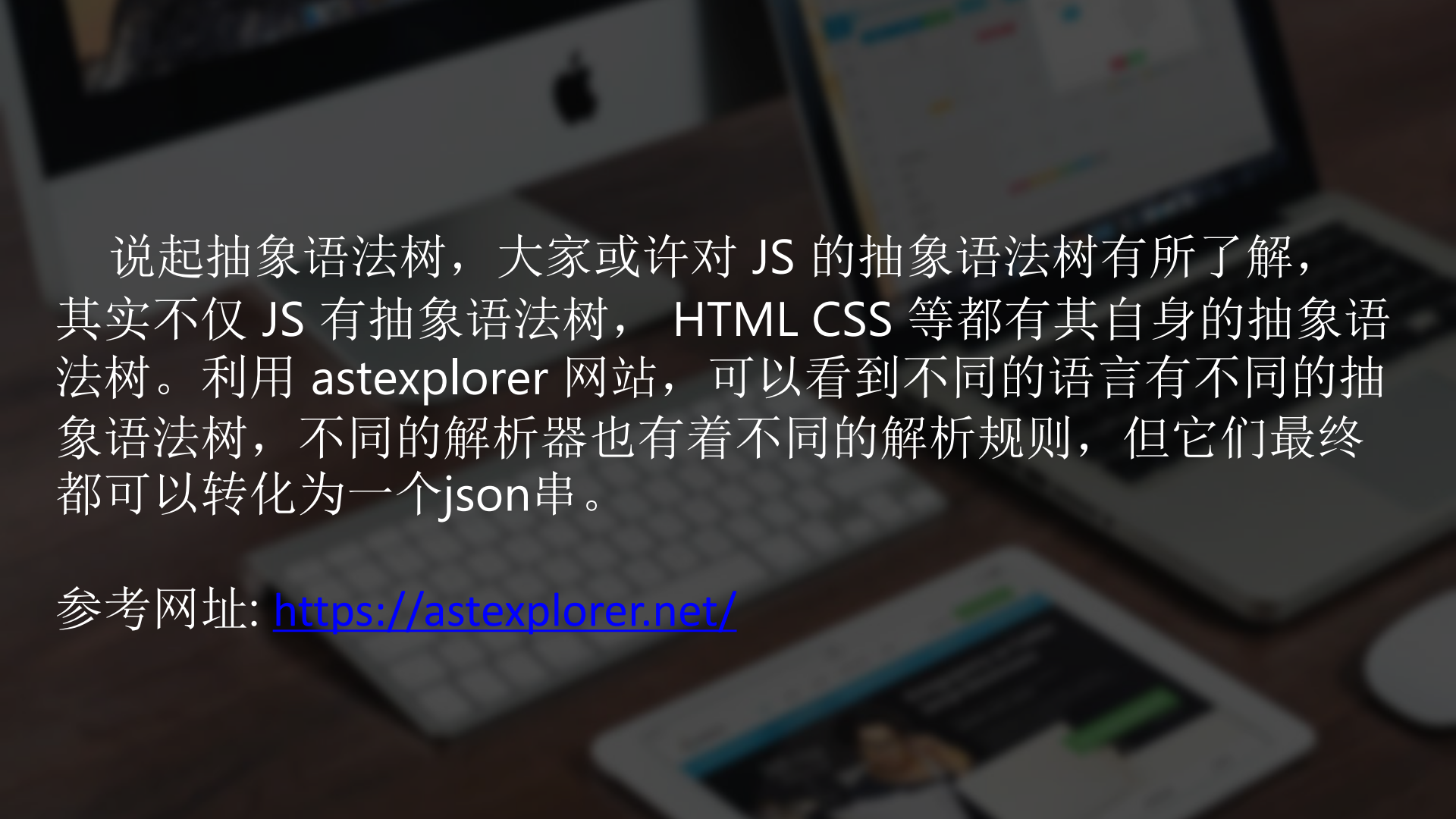
```
const cssObj = {
    table: {
        fontFamily: "'SimSun', 'Courier New', Courier, monospace",
        borderCollapse: 'collapse',
        borderSpacing: '0px',
    },
    td: {
        border: '1px solid #000',
        padding: 0,
    },
}

const cssString = util.convert(cssObj);
console.log(cssString)
```

```
.table {
  font-family: 'SimSun', 'Courier New', Courier, monospace;
  border-collapse: collapse;
  border-spacing: 0px;
}
.td {
  border: 1px solid #000;
  padding: 0;
}
```



前端接收到后端传递过来的 CSS 字符串时，需要转换为 JS 对象来进行后续处理使用。CSS 字符串转换为 JS 对象也可以利用字符串替换来进行转换。

这里介绍一个新的处理方式，利用 CSS 的抽象语法树来处理。



说起抽象语法树，大家或许对 JS 的抽象语法树有所了解，其实不仅 JS 有抽象语法树， HTML CSS 等都有其自身的抽象语法树。利用 astexplorer 网站，可以看到不同的语言有不同的抽象语法树，不同的解析器也有着不同的解析规则，但它们最终都可以转化为一个json串。

参考网址: https://astexplorer.net/

这里我们利用 rework 解析器来对 css 字符串进行处理

```css
#main {
    border: 1px solid black;
}

```

Tree JSON


```
{
  "type": "stylesheet",
  "stylesheet": {
    "rules": [
      {
        "type": "rule",
        "selectors": [
          "#main"
        ],
        "declarations": [
          {
            "type": "declaration",
            "property": "border",
            "value": "1px solid black",
            "position": {…}
          }
        ],
        "position": {…}
      }
    ],
    "parsingErrors": []
  }
}
```

通过上图可以看到转换的 css JSON字符串的样子，这里我们只关注 selectors 和 declarations，可以看出 selectors 即为 CSS 选择器的值， declarations 中的每个对象的 property 和 value 即为 CSS 的属性(property)和值(value)。转换部分代码如下，完整代码见附件。

```
if (rule.type === 'rule') {
  rule.declarations.forEach((declaration) => {
    if (declaration.type === 'declaration') {
      // 'align-items' ==> 'alignItems'
      const cleanProperty = cleanPropertyName(declaration.property);
      obj[cleanProperty] = declaration.value;
    }
  });
}
```


THANK YOU