测试

静态白盒测试

单元测试

单元测试描述：Picker Page 单元测试

单元测试描述：Calendar Page 单元测试

单元测试描述：Navigate Page 单元测试

单元测试描述：Picker Page 单元测试

测试脚本输出结果如下：

页面各模块测试（**集成测试**）

1. 初次进入小程序
2. 日历页面功能测试
3. 知识页面功能测试
4. 紧急呼叫功能测试
5. 紧急呼救页面功能测试

小程序兼容性测试

1. 界面布局和显示
2. 功能和交互

测试结论

小程序易用性测试——用户界面测试（UI Testing）

测试的不足之处

测试

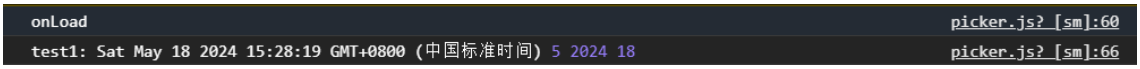
静态白盒测试

- 在对软件的静态白盒测试中，通过分析程序的代码来发现潜在的缺陷和问题，侧重于检查和审查程序的内部结构、设计和代码质量。

1.对 pages\picker\picker.js 文件代码进行静态白盒测试

```
//对定义的常量的值进行验证
const date = new Date();
const cur_year = date.getFullYear();
const cur_month = date.getMonth() + 1;
const day = date.getDate();
var nowday = cur_year + "-" + cur_month + "-" + day;
console.log('test1:',date,cur_month,cur_year,day);
```

测试结果如下：




```
//加入console打印数据，检查结果 以及在存储失败时，输出信息。
const date = e.currentTarget.dataset.date
const a = e.currentTarget.dataset.array
const a2 = e.currentTarget.dataset.arrayb
const age = e.currentTarget.dataset.arrayc
console.log('save_btn',date, a, a2, age)
try {
      //经期长度
      wx.setStorageSync('jinqi', a)
      //周期长度
      wx.setStorageSync('zhouqi', a2)
      //最近一次月经
      wx.setStorageSync('zuijinriqi', date)
      //年龄
      wx.setStorageSync('nianlin', age)
    } catch (e) {
      console.warn("Save failed.", e);
    // 输出详细错误信息
    console.error("Error details:", e.message);
    }
```

2.对 pages\calendar\calendar.js 文件代码进行静态白盒测试

```
  // 绘制当月天数占的格子
  calculateDays(year, month) {
    let days = [];
    //获得当月共有多少天
```

```
    const thisMonthDays = this.getThisMonthDays(year, month);
    for (let i = 1; i <= thisMonthDays; i++) {
      days.push(i);
    }
    console.log('calculateDays:',days);  //测试对应函数的输出日期是否符合实际
    //设置当月天数
    this.setData({
      days
    });
  },
```

测试的部分结果：

对应输出了每个月份获得的days的列表，并特殊检查了对应2月的天数。闰年为29天，其他年份为28天。以下是部分输出结果截图：


```
 //将当月所有的日期都根据条件判断一下，然后放入不同的数组中
    var MenstrualPeriod = [];  // 月经期
    var SafePeriod = [];  // 安全期
    var OvulationPeriod = [];  // 排卵期
    var OvulationDay = [];  // 排卵日
    // 排卵日--OvulationDay
    var MenstrualPeriod_test = [];  // 月经期       // 有可能有两组月经期，靠,例如上一个开始时间为5.1，周期为21，那么一月就有两组月经期
    var SafePeriod_test = [];  // 安全期
    var OvulationPeriod_test = [];  // 排卵期
    var OvulationDay_test = [];  // 排卵日，最多有两个，至少有一个,排卵日在两个月经开始时间中间

// 1、计算排卵日,这里没有处理排卵日的合法性
    // 显然，用到了lastDay的地方都要分类讨论
    // 本月有月经开始日
    if(lastDay!=0){
      // 排卵日1，上一个月经开始时间与本次月经开始时间的中间时间
      var OD1=Math.floor(-(MenstrualCycle)/2+lastDay)  // 记得取整
      // 排卵日2，本次月经开始时间与下一个月经开始时间的中间时间
      var OD2=Math.floor((MenstrualCycle)/2+lastDay)  // 记得取整
      // 排卵日3，下一次月经开始时间与下下个月月经开始时间的中间时间
      var OD3=Math.floor(3*(MenstrualCycle)/2+lastDay)
      console.log("OD1",OD1,OD2,OD3)
    }else{ // 本月没有月经开始日，那么只能依靠lastDay_continue了
      var OD1=Math.floor((MenstrualCycle)/2+lastDay_continue)-lastMonthsDays
      var OD2=Math.floor(3*(MenstrualCycle)/2+lastDay_continue)-lastMonthsDays
      var OD3=Math.floor(5*(MenstrualCycle)/2+lastDay_continue)-lastMonthsDays
      console.log("OD2",OD1,OD2,OD3)
    }

    // 将排卵日添加入数组中
    if(OD1>0&&OD1<=thisMonthDays){
```

```
      OvulationDay_test.push(OD1)
    }
    if(OD2>0&&OD2<=thisMonthDays){
      OvulationDay_test.push(OD2)
    }
    if(OD3>0&&OD3<=thisMonthDays){
      OvulationDay_test.push(OD3)
    }
    console.log("OvulationDay_test",OvulationDay_test)
    // 2、解决上个月遗留问题（排卵期）
    // 补充上个月的经期遗留问题
    if((lastDay_continue+MenstrualPeriodLength-
lastMonthsDays)>0&&lastDay_continue!=1000){
      for(let i = 1; i<lastDay_continue+MenstrualPeriodLength-
lastMonthsDays;i++){
        MenstrualPeriod_test.push(i)
      }
    }

    // 3、查看本月是否有月经开始日
    if(lastDay!=0){
    // 4、在有月经开始日的情况下，计算该开始日的月经期,若没有，转6
      for(let i = lastDay; i<lastDay+MenstrualPeriodLength;i++){
        if(i<=thisMonthDays){
          MenstrualPeriod_test.push(i)
        }else
          break
      }
    // 5、检查本月是否存在双月经开始日,存在则更新月经期
      if (lastDay+MenstrualCycle<=thisMonthDays){
        for(let i = lastDay+MenstrualCycle;
i<lastDay+MenstrualCycle+MenstrualPeriodLength;i++){
          if(i<=thisMonthDays){
            MenstrualPeriod_test.push(i)
          }
          else
            break
        }
      }
    }

    // 6、计算本月的排卵期（危险期），注意，当经期和排卵日相重叠时，以经期为主
    for(let i = OD2-5; i<OD2+5;i++){  // 排卵日2
      if(i!=OD2&&(!MenstrualPeriod_test.includes(i))&&i<=thisMonthDays){
        OvulationPeriod_test.push(i)
      }
    }
    for(let i = OD1-5; i<OD1+5;i++){  // 排卵日1
      if(i!=OD1&&i<=thisMonthDays&&i>0&&(!MenstrualPeriod_test.includes(i))){
        OvulationPeriod_test.push(i)
      }
    }
    for(let i = OD3-5; i<OD3+5;i++){  // 排卵日3
      if(i!=OD3&&i<=thisMonthDays&&i>0&&(!MenstrualPeriod_test.includes(i))){
        OvulationPeriod_test.push(i)
      }
```

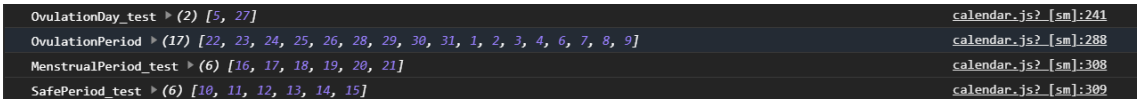
```
    }

    console.log("OvulationPeriod",OvulationPeriod_test)

    // 7、计算本月的安全期
    // 计算安全期
    var notSafe=[];
    for (const i of MenstrualPeriod_test) {
      notSafe.push(i)
    }
    for (const i of OvulationPeriod_test) {
      notSafe.push(i)
    }
    notSafe.push(OD1)
    notSafe.push(OD2)
    // console.log("notSafe",notSafe)
    for (let i=1; i<=thisMonthDays;i++){
      if (!notSafe.includes(i)){
        SafePeriod_test.push(i)
      }
    }

    console.log("MenstrualPeriod_test",MenstrualPeriod_test)
    console.log("SafePeriod_test",SafePeriod_test)
    console.log("OvulationPeriod_test",OvulationPeriod_test)
```

输出对应的计算的安全期、排卵期、危险期、经期四个列表的内容，检测对应的数据是否正确



单元测试

在 Node.js 和 npm 环境下，使用 Jest 测试框架编写和运行单元测试脚本，以确保代码的正确性和稳定性。

单元测试描述：Picker Page 单元测试

模块名称: Picker Page

测试目的:
确保 Picker Page 模块的核心方法和功能在不同条件下能够正确运行，包括页面加载时初始化数组和日期、从缓存加载数据、更新选择器的索引和日期、保存数据并切换页面。

测试范围:

1. **onLoad 方法**: 确保在页面加载时正确初始化数组和日期。
2. **onShow 方法**: 确保在页面显示时正确从缓存加载数据。
3. **bindPickerChange 方法**: 确保能够更新第一个选择器的索引值。
4. **bindPickerChange2 方法**: 确保能够更新第二个选择器的索引值。
5. **bindPickerChange3 方法**: 确保能够更新年龄选择器的索引值。
6. **bindDateChange 方法**: 确保能够更新日期选择器的值。
7. **save_btn 方法**: 确保能够保存数据并切换页面。

测试用例:

1. **页面加载测试 (onLoad)**:
   - **输入数据**: 无
   - **验证内容**: 在页面加载时，数组和日期是否正确初始化。
   - **期望结果**: `array` 数组长度为 8，`array2` 数组长度为 15，`array3` 数组长度为 99，`date` 格式正确。
2. **页面显示测试 (onShow)**:
   - **输入数据**: 无
   - **验证内容**: 在页面显示时，是否从缓存加载正确的数据。
   - **期望结果**: `index` 值为 4，`index2` 值为 7，`age` 值为 25，`date` 值为 '2023-04-17'。
3. **索引更新测试 (bindPickerChange)**:
   - **输入数据**: 选择器变更事件
   - **验证内容**: 更新第一个选择器的索引值。
   - **期望结果**: `index` 值为选择器变更事件中的值。
4. **索引更新测试 (bindPickerChange2)**:
   - **输入数据**: 选择器变更事件
   - **验证内容**: 更新第二个选择器的索引值。
   - **期望结果**: `index2` 值为选择器变更事件中的值。
5. **年龄索引更新测试 (bindPickerChange3)**:
   - **输入数据**: 年龄选择器变更事件
   - **验证内容**: 更新年龄选择器的索引值。
   - **期望结果**: `age` 值为年龄选择器变更事件中的值。
6. **日期更新测试 (bindDateChange)**:
   - **输入数据**: 日期选择器变更事件
   - **验证内容**: 更新日期选择器的值。
   - **期望结果**: `date` 值为日期选择器变更事件中的值。
7. **保存数据并切换页面测试 (save_btn)**:
   - **输入数据**: 保存按钮事件
   - **验证内容**: 验证是否正确保存数据到缓存并切换页面。
   - **期望结果**: 调用 `wx.setStorageSync` 方法保存数据，并调用 `wx.switchTab` 方法切换页面。

测试输入数据:

1. **模拟 wx 对象及其方法**:
   - `wx.getStorageSync` 模拟返回的数据：

```
{
  jinqi: 7,
  zhouqi: 28,
  zuijinriqi: '2023-04-17',
  jinqi_index: 4,
  zhouqi_index: 7,
  nianlin_index: 25
}
```

- `wx.setStorageSync` 和 `wx.switchTab` 方法使用 Jest 进行模拟。

2. **初始化 page 对象**:
   - 初始化数据:

```
{
  array: [],
  array2: [],
  array3: [],
  date: '2016-09-01',
  index: 5,
  index2: 28,
  tocalendar: 0,
  age: 0
}
```

单元测试描述：Calendar Page 单元测试

模块名称: calendar Page

测试目的:
确保 `Calendar Page` 模块的核心方法和功能在不同条件下能够正确运行，包括数据初始化、月份切换、天数计算、空格子计算、系统信息获取，以及根据用户数据计算不同的生理周期阶段（如月经期、安全期、排卵期、排卵日）。

测试范围:

1. **onShow 方法**: 确保在页面显示时正确初始化数据。
2. **handleCalendar 方法**: 确保在切换月份时正确更新数据。
3. **calculateDays 方法**: 确保能够正确计算每个月的天数。
4. **calculateEmptyGrids 方法**: 确保能够正确计算每个月前置空格子的数量。
5. **getSystemInfo 方法**: 确保能够正确获取系统信息并设置数据。
6. **viewyue 方法**: 确保能够根据用户输入的数据正确计算生理周期的各个阶段。

测试用例:

1. **初始化测试 (onShow)**:
   - 验证在页面显示时，生理周期的各个阶段数组（`yue`, `anquan`, `weixian`, `pailuanri`）是否正确初始化并有数据。
   - **期望结果**: 所有数组的长度应大于 0。
2. **月份切换测试 (handleCalendar)**:
   - 验证切换到下一个月时，当前月份是否正确增加一个月并更新数据。
   - 验证切换到上一个月时，当前月份是否正确减少一个月并更新数据。

    - **期望结果**: `cur_month` 值正确变化并且数据更新。
3. **天数计算测试 (calculateDays)**:
    - 验证计算特定年份和月份的天数是否正确，例如 2024 年 5 月应有 31 天。
    - **期望结果**: `days` 数组长度应为 31。
4. **空格子计算测试 (calculateEmptyGrids)**:
    - 验证计算特定年份和月份前置空格子的数量是否正确，例如 2024 年 5 月应有 3 个空格子。
    - **期望结果**: `empytGrids` 数组长度应为 3。
5. **系统信息获取测试 (getSystemInfo)**:
    - 验证系统信息是否正确获取并设置，例如窗口宽度应为 375，滚动视图高度应为 1334。
    - **期望结果**: `canvasewidth` 应为 375， `scrollViewHeight` 应为 1334。
6. **生理周期阶段计算测试 (viewyue)**:
    - 验证在特定年月（如 2024 年 5 月），根据用户输入的数据是否正确计算并设置生理周期的各个阶段（`yue`, `anquan`, `weixian`, `pailuanri`）。
    - **期望结果**: 根据预期值验证 `yue`, `anquan`, `weixian`, `pailuanri` 数组的内容。

测试输入数据:

- 模拟用户数据:

```
global.wx = {
  getStorageSync: (key) => {
    const data = {
      jinqi: 7,
      zhouqi: 28,
      zuijinriqi: '2023-04-17',
    };
    return data[key];
  },
  setStorageSync: jest.fn(),
  createCanvasContext: jest.fn(() => ({
    arc: jest.fn(),
    setFillStyle: jest.fn(),
    fill: jest.fn(),
    setFontSize: jest.fn(),
    fillText: jest.fn(),
    draw: jest.fn(),
  })),
  getSystemInfoSync: () => ({
    windowHeight: 667,
    windowWidth: 375,
    pixelRatio: 2,
  }),
};
```

详细测试脚本见文件（./test/calendar.test.js）

单元测试描述：Navigate Page 单元测试

模块名称: Navigate Page

测试目的:
确保 `Navigate Page` 模块的核心方法 `navigateToLink` 在不同条件下能够正确运行，验证页面跳转的功能。

测试范围:

1. **navigateToLink 方法**: 确保在调用时能够正确跳转到目标页面。

测试用例:

1. **页面跳转测试 (navigateToLink)**:
    - 验证在调用 `navigateToLink` 方法时，能够根据传入的事件对象中的目标 URL 正确跳转到相应的页面。
    - **期望结果**: 调用 `wx.navigateTo` 并传入正确的目标 URL。

测试输入数据:

- 模拟的事件对象：

```
const event = {
  currentTarget: {
    dataset: {
      url: 'examplePage'
    }
  }
};
```

测试用例描述:

1. **页面跳转测试 (navigateToLink)**:
    - **输入**: 模拟的事件对象 `event`，包含目标页面的 URL `examplePage`。
    - **操作**: 调用 `page.navigateToLink(event)`。
    - **预期输出**: 验证 `wx.navigateTo` 方法被调用且传入的 URL 为 `/pages/examplePage/examplePage`。

期望结果:

- `wx.navigateTo` 方法被正确调用，并且参数中包含预期的目标 URL。

详细测试脚本见文件（./test/list.test.js）

单元测试描述：Picker Page 单元测试

模块名称: Picker Page

测试目的:
确保 Picker Page 模块的核心方法和功能在不同条件下能够正确运行，包括页面加载时初始化医院信息、切换显示状态、选择地区和医院、更新索引和输入值、验证和保存输入值，以及从存储中获取数据并拨打电话。

测试范围:

1. **onLoad 方法**: 确保在页面加载时正确初始化医院信息。
2. **toggleDistricts 方法**: 确保能够切换地区选择器的显示状态。
3. **selectDistrict 方法**: 确保能够选择地区并更新对应的医院列表。
4. **selectHospital 方法**: 确保能够选择医院并更新选中的医院。
5. **bindPickerChange 方法**: 确保能够更新第一个选择器的索引值。
6. **bindPickerChange2 方法**: 确保能够更新第二个选择器的索引值。
7. **bindinput 方法**: 确保能够更新输入的电话号码。
8. **submit 方法**: 确保能够验证输入的电话号码并保存数据。
9. **sos 方法**: 确保能够从存储中获取电话号码并拨打电话。

测试用例:

1. **页面加载测试 (onLoad)**:
   - **输入数据**: 无
   - **验证内容**: 在页面加载时，医院的地区列表和医院信息是否正确初始化。
   - **期望结果**: `districts` 数组长度大于 0，`hospitals` 对象与预期值相同。
2. **切换显示状态测试 (toggleDdistricts)**:
   - **输入数据**: 无
   - **验证内容**: 切换显示地区选择器的状态。
   - **期望结果**: `showDistricts` 值应为切换前的相反值。
3. **选择地区测试 (selectDistrict)**:
   - **输入数据**: 选择的地区事件
   - **验证内容**: 选择地区后，选中的地区和对应的医院列表是否正确更新。
   - **期望结果**: `selectedDistrict` 值应为选择的地区，`districtHospitals` 值应为该地区对应的医院列表。
4. **选择医院测试 (selectHospital)**:
   - **输入数据**: 选择的医院事件
   - **验证内容**: 选择医院后，选中的医院是否正确更新。
   - **期望结果**: `selectedHospital` 值应为选择的医院。
5. **索引更新测试 (bindPickerChange)**:
   - **输入数据**: 选择器变更事件
   - **验证内容**: 更新第一个选择器的索引值。
   - **期望结果**: `index` 值应为选择器变更事件中的值。
6. **索引更新测试 (bindPickerChange2)**:
   - **输入数据**: 选择器变更事件
   - **验证内容**: 更新第二个选择器的索引值。
   - **期望结果**: `index2` 值应为选择器变更事件中的值。
7. **输入值更新测试 (bindinput)**:
   - **输入数据**: 输入值变更事件
   - **验证内容**: 更新输入的电话号码。

- **期望结果**: `ipt.tel` 值应为输入的电话号码。

8. **保存输入值测试 (submit)**:
   - **输入数据**: 提交事件
   - **验证内容**: 验证输入的电话号码并保存数据。
   - **期望结果**:
     - 当 `ipt.tel` 为空时，调用 `wx.showToast` 显示提示信息。
     - 当 `ipt.tel` 不为空时，调用 `wx.setStorageSync` 保存数据，并调用 `wx.showToast` 显示成功信息。
9. **拨打电话测试 (sos)**:
   - **输入数据**: sos 事件
   - **验证内容**: 从存储中获取电话号码并拨打电话。
   - **期望结果**: 调用 `wx.getStorage` 获取电话号码，并调用 `wx.makePhoneCall` 拨打电话。

测试输入数据:

1. **模拟 wx 对象及其方法**:
   - `wx.getStorageSync` 模拟返回的数据：

     ```
     {
       ipt: { tel: '12345678901' }
     }
     ```

   - `wx.setStorageSync`、`wx.showToast`、`wx.getStorage` 和 `wx.makePhoneCall` 方法使用 Jest 进行模拟。
2. **初始化 page 对象**:
   - 初始化数据：

     ```
     {
       array: [],
       array2: [],
       date: '2016-09-01',
       index: 5,
       index2: 28,
       tocalendar: 0,
       ipt: { tel: '' },
       districts: Object.keys(hospitals),
       hospitals: hospitals,
       selectedDistrict: '',
       showDistricts: false,
       districtHospitals: [],
       selectedHospital: null
     }
     ```

详细测试脚本见文件（./test/call.test.js）

测试脚本输出结果如下：

```
> crimson-diary@1.0.0 test
> jest

 PASS  test/calendar.test.js
 PASS  test/call.test.js
  ● Console

    console.log
      拨打电话成功!

      at Object.log [as success] (test/call.test.js:124:21)

 PASS  test/picker.test.js
  ● Console

    console.log
      onLoad

      at Object.log [as onLoad] (test/picker.test.js:90:13)

    console.log
      show data 4 7 2023-04-17 25

      at Object.log [as onShow] (test/picker.test.js:66:15)

    console.log
      save_btn 2023-05-01 4 7 25

      at Object.log [as save_btn] (test/picker.test.js:116:13)

 PASS  test/list.test.js

Test Suites: 4 passed, 4 total
Tests:       25 passed, 25 total
Snapshots:   0 total
Time:        1.214 s
Ran all test suites.
```

页面各模块测试（集成测试 ）

1. 初次进入小程序

测试步骤：

- 进入小程序，设置经期周期、年龄、经期长度和最近一次经期时间。
- s点击“保存”按钮。

预期结果：

- 页面应成功跳转到相应的页面。

实际结果：

- 页面成功跳转至预期页面，数据保存正常。

2. 日历页面功能测试

测试步骤：

- 打开日历页面。
- 查看日历上标记的安全期、经期、危险期和排卵期。
- 点击不同的月份进行跳转。

预期结果：

- 日历应准确显示安全期、经期、危险期和排卵期。
- 点击月份跳转后，日历显示应无误。

实际结果：

- 日历页面正常显示安全期、经期、危险期和排卵期。
- 点击月份跳转后，日历显示正确，功能正常。

3. 知识页面功能测试

测试步骤：

- 点击小程序中的知识页面链接。
- 查看页面内容是否正确加载和显示。

预期结果：

- 页面应成功跳转至知识页面，并且内容应正确显示。

实际结果：

- 知识页面成功跳转，所有内容正确显示且页面加载流畅。

4. 紧急呼叫功能测试

测试步骤：

- 在紧急呼叫页面输入有效的手机号。
- 点击“拨号”按钮。

预期结果：

- 系统应正常发起拨号请求，能够成功拨出电话。

实际结果：

- 拨号功能正常，点击“拨号”按钮后成功发起电话呼叫。

5. 紧急呼救页面功能测试

测试步骤：

- 打开紧急呼救页面。
- 查看不同地区对应医院的电话号码和地址信息。

预期结果：

- 页面应准确显示不同地区的医院信息，包括电话号码和地址。

实际结果：

- 紧急呼救页面显示正常，不同地区医院的电话号码和地址信息准确无误，信息展示全面且清晰。

此次测试覆盖了小程序的主要功能模块，包括初次设置和数据保存、日历功能、知识页面、紧急呼叫功能以及紧急呼救信息展示。所有测试均通过，验证了小程序在各个功能模块上的正确性和稳定性。用户体验良好，功能操作顺畅，系统运行无明显漏洞或错误。

小程序兼容性测试

测试工具: 微信开发者工具、各小组成员的手机

测试内容:

1. 界面布局和显示
2. 功能和交互

1. 界面布局和显示

分辨率和屏幕尺寸:

- **测试方法**：在微信开发者工具中模拟不同分辨率和屏幕尺寸（如高清屏、全面屏等），并在各小组成员的手机上实际运行小程序。
- **测试结果**：在所有测试设备上，小程序的布局和显示均正常，没有出现元素重叠、错位等问题。界面自适应不同分辨率，显示效果良好。

字体大小和样式:

- **测试方法**：检查不同设备上的文本可读性，包括字体大小和样式的一致性。
- **测试结果**：在不同机型上，文本的字体大小和样式一致，可读性良好，没有出现字体过大或过小的问题。

图片和图标:

- **测试方法**：确认图片和图标在不同屏幕分辨率上是否清晰可见，比例是否正确。
- **测试结果**：所有设备上的图片和图标均清晰可见，比例正确，没有出现模糊或失真的情况。

响应式设计:

- **测试方法**：在微信开发者工具中切换横屏和竖屏模式，并在实际设备上测试响应式设计效果。
- **测试结果**：小程序在横屏和竖屏模式下均能正常显示，布局自适应变化，用户体验良好。

2. 功能和交互

触摸和手势操作:

- **测试方法**：在各小组成员的手机上测试点击、滑动、长按等触摸操作。
- **测试结果**：所有触摸操作均能正常响应，没有出现无法点击或手势操作失效的问题。

组件兼容性:

- **测试方法**：检查不同 UI 组件（如按钮、输入框、下拉菜单等）在各个机型上的表现是否一致。
- **测试结果**：所有 UI 组件在不同机型上表现一致，按钮点击、输入框输入、下拉菜单选择等操作均正常。

页面跳转和导航:

- **测试方法**：验证页面跳转和导航的流畅性和正确性，包括页面间的链接和返回操作。
- **测试结果**：页面跳转和导航流畅，所有页面间的链接和返回操作均能正常运行，没有出现卡顿或跳转错误的问题。

表单输入:

- **测试方法**：测试表单输入和提交功能，包括键盘弹出和收起的行为。

- **测试结果**：表单输入和提交功能正常，键盘弹出和收起行为符合预期，没有出现输入内容丢失或提交失败的问题。

测试结论

经过利用微信开发者工具和小组成员手机的全面测试，小程序在不同分辨率、屏幕尺寸、和操作系统版本的设备上均能正常运行，界面布局、功能和交互均表现良好，满足兼容性要求。

小程序易用性测试——用户界面测试（UI Testing）

- **页面导航**：
  - 小程序的设计简洁直观，底部导航栏清晰列出了四个主要功能，每个功能都有相关的文字描述，用户能够快速了解和访问各个功能模块。导航栏的布局合理，易于用户操作。
- **不同层级跳转功能**：
  - 紧急呼叫页面
    - 点击不同地区的选项后，能够显示该地区不同医院的详细信息。页面层级之间的跳转和返回操作流畅，用户体验友好。
  - 知识模块
    - 在知识页面，点击相应按钮后，能够顺利跳转到对应的信息页面。按键设计简洁明了，页面切换迅速，符合用户使用习惯。
  - 功能入口
    - 小程序实现的功能涵盖了用户常用的业务需求，并且各功能入口设计合理，与小程序整体的简洁性设计相符，用户可以轻松找到所需功能。

总结：

- **导航**：小程序的底部导航栏设计合理，功能介绍明确直观，便于用户操作。
- **跳转逻辑**：层级跳转逻辑清晰，操作简便，用户能够快速返回和进入不同页面。
- **界面设计**：整体界面设计简洁，与功能需求匹配，符合用户使用习惯。

此次易用性测试验证了小程序在页面导航、层级跳转和功能入口设计上的合理性和用户友好性。用户可以轻松上手，顺利完成各项操作，体现了良好的用户体验和界面设计。

测试的不足之处

受限于小程序并没有完整实现相关的用户登陆以及后端的数据库、数据管理等内容，以上关于性能测试和安全性测试的内容并没有作深入测试。故在本次大作业的测试部分被忽略。