《软件测试课程设计》讲稿（第 8 章）

第 10 周 · 2 学时

广东金融学院 计算机学院 | 周宇文

讲稿可照读版 | 由课件逐页撰写

第 8 章　软件本地化测试

第 10 周 · 2 学时（90 分钟） | 教材：朱少民《软件测试方法和技术（第 4 版）》

《软件测试课程设计》第 8 章 讲稿（第 10 周 · 2 学时 90 分钟）

> 课程：软件测试课程设计（154442008） | 广东金融学院 计算机学院 | 专业选修
>
> 章节：第 8 章 软件本地化测试 —— 同一套代码，能不能在另一个语言和地区"像本地产品一样好用"
>
> 学时：**2 学时（90 分钟）**
>
> 教材：朱少民《软件测试方法和技术（第 4 版）》第 2 篇 软件测试技术
>
> 配套课件：逐页还原版 **46 页**（`lecture8/`，云端 http://111.231.0.226:2345/lecture8/index.html）
>
> 教案对应：第 10 次课「软件本地化测试」
>
> 本讲重点：国际化（i18n）与本地化（l10n）的区别；本地化测试的测试项（翻译质量、界面布局与截断、日期时间与货币格式、排序与检索、输入法与字符集、图片图标、法规与习俗）；测试组织（单语言/多语言环境、伪本地化、翻译验证与回归）
>
> 本讲难点：为什么"能显示中文"不等于"本地化做对了"；伪本地化如何在不翻译的情况下提前暴露布局问题；字符集与编码问题的定位思路

> **本讲稿为可照读完整版**：每页含【口播】【提问＋预设回答】【易错点/考点】【过渡】；带 ★ 的页为可压缩页。

一、教学目标

维度	目标（课后可检验）
知识	① 能区分国际化与本地化，并说明各自发生在哪个阶段、由谁负责；② 能列出本地化测试至少 8 类测试项；③ 能解释伪本地化的原理与价值；④ 能说出编码/字符集问题的常见现象与排查方向；⑤ 能说明本地化回归测试的范围怎么定
能力	① 能为一个已上线的中文系统列出"出海到英文与阿拉伯语市场"的本地化测试清单；② 能设计一套伪本地化验证方案；③ 能判断一个界面问题是翻译问题、布局问题还是编码问题
素质（思政）	① 由编码/乱码问题体会"细节决定成败"；② 由习俗与法规禁忌体会尊重文化差异与合规意识；③ 由翻译质量体会对用户负责；④ 由测试组织体会流程与严谨

本讲要建立的一个总印象：本地化测试测的不是"翻译对不对"，而是"**这个产品在这个市场里，像不像一个本地产品**"——语言、格式、习惯、法规，四样都要对。

二、时间分配总表

> **容量提示**：本章课件 **46 页**，讲稿是素材全量版（口播约 2 万字，≈85 分钟），课时 90 分钟，**基本可以按页讲完**；仍建议把纯图示页与工具页快速带过。

时段	页码	内容	分钟	备注
第 1 节	8.1–8.12	开场、i18n 与 l10n、本地化测试的定位与测试项	26	**测试项清单必讲**

第 1 节	8.13–8.26	界面布局与截断、格式本地化（日期/时间/货币/数字）、排序与检索	24	现场找截断例子
——	——	**课间休息 10 分钟**	——	——
第 2 节	8.27–8.38	字符集与编码、输入法、图片图标、法规与习俗	20	**编码问题必讲**
第 2 节	8.39–8.46	测试组织与执行、伪本地化、回归、小结与作业	15	伪本地化当堂设计

> **压缩优先级**：先压缩工具截图页与流程图示页 → 再压缩具体语种细节 → 最后才压缩"测试项清单""编码问题""伪本地化"三处。

三、逐页讲稿

> 每页第一段是可直接朗读的口播；其后是提问与预设回答、易错点/考点、过渡语。
>
> 页码锚点与课件页一一对应（8.1–8.46）：http://111.231.0.226:2345/lecture8/index.html

【8.1】软件测试方法和技术（1 分钟）

> 换个问法开头。前面七章反复回答的是同一件事：这个软件对不对。对不对背后有个没人说出口的前提——它是给中国人用的。界面写中文，日期是年-月-日，钱是人民币，姓名姓在前、名在后。

>

> 这个前提一撤，问题就换了。同一个安装包，装到阿拉伯语用户的机器上，界面得从右往左排。装到德语机器上，同一句提示会长出一截，按钮装不下。装到美国机器上，2/3/2009 说不清是二月三日还是三月二日。这几件事都不算功能错，算换个地方还能不能用、用得顺不顺。软件能不能卖出去，常常先卡在这一关，功能好不好反倒排在后面。

>

> 于是这一章要回答三件事。i18n 和 l10n 差在哪。本地化测试到底要查什么。这些活怎么组织和执行。

>

> 前面几章各管一段：单元测一个方法对不对，集成测拼起来通不通，系统功能测整体，性能和安全各有各的查法。这些活都在同一个前提下展开，就是软件说中文、给中国人用。前提一换，一半用例要重新想。

>

> 课件的四节我先报一遍：8.1 什么是软件本地化，8.2 翻译验证，8.3 本地化测试的技术问题，8.4 本地化的功能测试。定义和编码那一块占的时间最长，技术问题那一块最像干货，几乎每一条都能直接落成一条用例。

>

> 带走一个问题，边听边想：你的软件明天要卖到德国和沙特，手上这些用例，有几条还能直接跑？一条都数不出来，这一章你就得跟着走。

提问：一个在国内用得好好的软件，直接拿到国外卖，你猜第一批投诉会集中在哪儿？

> **预设回答：**

- 答"翻译错了、翻得怪"——这是最直接的一类，可它只占问题的一部分，后面还有格式、布局、文化三类。

- 答"界面乱码、日期看不懂、钱的写法不对"——很准，这三条正好对应字符集和数据格式两个技术板块。

- 答"翻成英文就能卖了"——这是最典型的错答。翻译只是本地化六项工作里的一项，代码没做国际化改造，日期和货币还是硬编码，翻完照样用不了。

- 收口：顺序摆在这儿，先有国际化能力，才谈得上本地化质量。

易错点 / 考点：① 两条主线必须分开：i18n 是让软件具备被本地化的能力，l10n 是把它变成某个地区的版本；② 本地化不等于翻译，这个判断全章反复考；③ 判断题陷阱"本地化测试只在测试阶段做"——错，成败从设计、编码就已经埋下了。

> **过渡：**定义先放一放，先看几个你我都碰过的生活场面。

【8.2】国际化生活的体验（3 分钟）

出过国的同学回来讲感受，头一句往往不是"听不懂"。是那些看着一样、用起来不一样的东西。

>

> 插头插不进插座。温度说的是华氏度，你说 30 度他以为要冻死。距离用英里，重量用磅，纸是 Letter 不是

> A4。星期一算一周的第一天，还是星期日算第一天，两边也不一样。这些没对错，是约定不同。软件要卖过去，就得把这些约定一条条对齐。

>

> 第二个场面，2/3/2009 是哪一天。停三秒。美国人写月/日/年，是二月三日；欧洲和很多亚洲国家写日/月/年，是三月二日。同一串数字两个答案，而且两个都对。这是本地化里最阴的一类缺陷：程序不报错、不崩、数据也存进去了，只是含义错了。用户看到的日期差一个月，在金融系统里就是事故。

>

> 第三个，George Bush 属于哪个家族。问的是姓名和称谓。中文姓在前，英文名在前；中文叫布什先生，英文叫 Mr. Bush，称谓的位置反过来。还有音译：Bush 译成布什，同一家族的译法要一致，不一致，检索、排序、去重全乱。客户档案里一条姓名规则错了，几十万条数据要重洗。

>

> 第四个，邮件打开一片乱码。写的时候用一种编码，读的时候用另一种，就成了这样。

>

> 四个场面不是随手凑的。第一个落到度量衡和排版，第二个落到数据格式，第三个落到姓名和排序，第四个落到字符集。后面的章节就照这四条往下走。

>

> 四件事收一句：本地化测试要防的不是程序算错，是程序按别人的规矩算对了、按你的规矩算错了。

> **提问：**2/3/2009 这个日期，如果让你在程序里存下来，怎么保证不出歧义？

> **预设回答：**

- 答"按美国习惯存，月/日/年"——那欧洲用户填进来的日期就会被读错，方向反了。

- 答"用四位年-两位月-两位日"——对，这是国际上最不容易误读的写法，年放最前面，因为年的位数最多、最不会被当成日或月。

- 答"加个下拉框让用户选格式"——能缓解输入歧义，可存储层还是字符串的话，导出、排序、跨系统交换又出问题。

- 收口：可靠的做法是存储用无歧义格式、显示用本地格式，两个层面分开管。

易错点 / 考点： ① 日期歧义是案例题常客，答题要点是"显示本地化、存储标准化"；② 姓名问题有三个子项：顺序、称谓位置、音译一致性，只答"顺序"会丢分；③ 乱码的根因一句话，编码和解码用的字符集不一致，不是"字库坏了"。

过渡： 生活里的差异说完了，再看同一个软件在两地长什么样。

【8.3】具体例子（2 分钟）

两张图并排放着，同一个软件、同一份数据，左边中文简体在北京，右边英文在纽约。看的时候带个任务：找不同，并且给每处不同归个类。

>

第一类，文字本身。界面上的词换了语言。这一类最好发现，也最容易被当成全部。

>

第二类，格式。日期怎么写，时间是 12 小时制还是 24 小时制，钱用什么符号，千分位用什么，小数点用点还是用逗号。

>

第三类，排列。年月日还是月日年，姓在前还是名在前，地址从大到小还是从小到大。

>

第四类，度量。长度、重量、温度、纸张。

>

四类里只有第一类会被用户当场发现并且投诉。后三类最贵，因为它们不报错。举个具体的：一份报表在北京导出是 1,234.56，到德语环境可能变成 1.234,56，逗号和点的含义整个反过来。下游再有个程序去解析这串字符，一千二百三十四就被读成一点二三。

>

所以看这组对照，练一个动作：每看到一处差异就问一句，它停在显示层，还是会流进数据层？停在显示层的，改起来便宜。流进数据库、传给下游的，就是缺陷，而且常常发布以后才炸。

>

> 还有个更隐蔽的地方：同一处差异，对不同用户的伤害不一样。日期顺序错了，本地用户扫一眼就懂，外地用户看得一头雾水。这类问题报上来的缺陷特别少，因为它不疼不痒，用户只会觉得这软件别扭。

>

> 方法上也提一句，别拿眼睛扫。列成清单逐项过：日期、时间、数字、货币、度量、姓名、地址、电话、排序。这张清单后面讲数据格式那一节还要展开。

提问：这组图里的差异，哪些改翻译文件就能解决，哪些解决不了？

预设回答：

- 答"文字变了的那种，改翻译文件就行"——对，这是资源文件层面的活。

- 答"日期和钱的写法也能在翻译文件里改"——错。格式是代码按区域设置算出来的，翻译文件里改不了；代码里写死了格式串，就得改代码。

- 答"得看它是怎么实现的"——这个回答最专业。判断依据就是：这个值是格式化出来的，还是拼字符串拼出来的。拼出来的，本地化必出问题。

- 收口：所以看到一个可疑显示，第一反应是问"它是怎么生成的"，不是问"翻译得对不对"。

易错点 / 考点：① 四类差异（文字、格式、排列、度量）要能分类举例；② 常考判断"界面文字都翻译了，本地化就完成了"——错，格式类问题一条没解决；③ 显示层和数据层这条分界，是判断缺陷严重度的实用依据。

过渡：中文和英文好歹都从左往右读，下一个例子把方向也翻过来。

【8.4】示例—阿拉伯语（2 分钟）

> 先看那排字的位置，别管它写的是什么。阿拉伯语从右往左读，行业里叫 RTL。这组图要你看出来的不是"这些字我不认识"，是整个界面的空间逻辑被镜像了。

>

> 镜像在哪几处？阅读方向反过来，文字从右边起头，段落右对齐。控件顺序反过来，原来左边的导航跑到右边，滚动条跑到左边，进度条的起点落在右端。

>

> 成对的符号要翻。左括号右括号在视觉上对调，问号、逗号的位置也跟着挪。

>

> 数字也有两套写法。阿拉伯文有自己的数字字符，跟英文那十个数字不是一个码位，同一份报表在两种字体下宽度都不一样。测的时候专门看一眼，数字是不是本地写法，还是被硬转成了英文数字。

>

> 数字和混排更麻烦。阿拉伯文里夹英文单词、夹数字，一段话里两种方向并存，边界在哪、这串数字按哪边读，要靠双向文本的算法去判。输入框里更细：光标往哪边跳，选区从哪头拉，两边不一致，用户会以为自己打错字。

>

> 这件事对测试为什么重要？因为它翻译解决不了。你把界面文字全换成阿拉伯语，布局代码还是按左对齐、左边距、控件从左往右排写死的，翻完就是灾难：文字压在图标上，按钮顺序颠倒，进度条反着走。这类问题在中文和英文环境里永远暴露不出来，只有真切到 RTL 才会出现。

>

> 业界的做法是在做国际化改造时就把布局写成跟着语言方向走。要写起始边、结束边的地方，别写死左右。测试也一样，别只测中文和英文，至少挑一个 RTL 语言过一遍，把镜像逻辑验掉。

>

> 判断一个界面有没有真支持 RTL，最省事的办法是看它有没有把"左/右"换成"起始/结束"这种跟方向无关的说法。

> **提问：**界面文字全换成阿拉伯语，布局代码一行不改，你估计最先崩的是哪儿？

> **预设回答：**

- 答"文字和图标会重叠"——很可能，文字变长、方向反了，原来的留白全对不上。

- 答"滚动条跑到左边，用户找不到"——对，控件镜像没做就是这样。

- 答"字体能显示就行，大概没大问题"——典型错答。方向、对齐、控件顺序都要单独处理，光能显示字符离"能用"差得远。

- 收口：所以看一个产品有没有真做 RTL，先找它有没有"起始/结束"这种表达。

> **易错点 / 考点：** ① RTL 的四个镜像点要能说出来；② 常考判断"支持双字节字符就等于支持阿拉伯语"——错，阿拉伯语是字符集加方向两件事；③ 简答题写术语 RTL／LTR 比写"从右往左"更得分。

> **过渡：** 场景看完了，回到教材的顺序，把这一章的地图铺开。

【8.5】8.1 什么是软件本地化（1 分钟）

> 四节的名字我按顺序点一遍，你记结构，细节后面逐个来。

>

> 8.1 什么是软件本地化。定义和基础都在这儿：i18n 和 l10n 什么关系，字符集是什么，国际化有哪些标准，本地化按什么步骤做，本地化测试包含哪几类工作。这一节占的时间最长。

>

> 8.2 翻译验证。翻译是本地化里最显眼的一项，可"翻完了"和"翻对了"是两件事。这一节讲怎么验翻译，以及翻译之外文化那一层要改什么。翻完了机器能查，翻对了只有人能判。

>

> 8.3 本地化测试的技术问题。三块：数据格式、页面显示和布局、配置与兼容性。这一节最实用，几乎每条都能变成一条用例。

>

> 8.4 本地化的功能测试。把本地化版本当成一个完整软件去测：集成、索引排序、联机文档，还有本土测试和翻译验证测试这两种组织形式。

>

> 四节读下来是一条递进：8.1 讲概念，8.2 讲内容，8.3 讲技术，8.4 讲执行。课程设计里可以先拿 8.3 的清单自查一遍，再按 8.4 组织一轮实测。

>

> 这些内容有一条共同的主线：本地化测试查的不是程序算得对不对，是程序在别人的规矩下还对不对。所以

> 既要懂测试，也要懂一点别人的规矩。课程设计要是做双语界面，8.1 和 8.3 这两节至少各留一页笔记。

>

> 四块里哪一块最难自动化？格式、编码、布局能写脚本扫，可"这个译法在这个文化里合不合适"，脚本给不出答案。所以本地化测试的典型配置是工具做筛查、人做判断，跟性能测试一个道理。

提问：这四块里，你觉得哪一块最难自动化、最依赖人？

> **预设回答：**

- 答"8.2 翻译验证"——有道理，语言好不好要人判断，机器只能查术语一致性。

- 答"8.3 技术问题"——也对，很多是"看着别扭"，判据难写。

- 答"都能自动化，写脚本扫一遍"——错。检测可以自动化，判断不行。

- 收口：工具做筛查、人做判断，这是本地化测试的常规配置。

易错点 / 考点：① 四块的名称与顺序；② 8.3 和 8.4 容易混：8.3 是技术问题清单，8.4 是把本地化版本当完整软件测的组织方式；③ 简答题问"本地化测试包括哪些内容"，标准答案是 8.1.5 那六类，别拿 8.3 的三块去答。

> **过渡：**先看 8.1 内部的路线，五小节，一层比一层落地。

【8.6】8.1 什么是软件本地化（1 分钟）

8.1 这一节里有五小节，一层比一层落地，我给你画条线。

>

> 8.1.1 软件本地化与国际化。先把 i18n 和 l10n 掰开。这两个词全章最容易混，也是必考的一对。

>

> 8.1.2 字符集问题。从 ASCII 讲到 Unicode、UTF-8。为什么放在这个位置？因为乱码是本地化故障里出现频率最高、用户最不能忍的一类，而它的根子全在字符集。编码不对，翻译再准也没用。

>

> 8.1.3 软件国际化标准。五条设计要求：切换语言的机制、与语言无关的输入输出接口、资源文件国际化、

> 包容本地数据格式。这五条是"一个软件够不够格被本地化"的检查表。名字先对齐，内容后面逐条展开。

>

> 8.1.4 软件本地化基本步骤。八步：配置管理、术语表、提取、翻译、检入、编译、调界面、再测试。它回答的是这件事怎么组织。

>

> 8.1.5 软件本地化测试。落到六类工作：功能性、翻译、可用性、兼容性、文化宗教、手册验证。

>

> 顺序别背反。有人一上来就问本地化怎么做，那是 8.1.4 的事；有人一上来就要测，那是 8.1.5。前面两个小节没弄清，后面两个只能照着做，说不出为什么。

>

> 串起来是一条线：概念分清、底层看编码、设计看标准、执行看流程、验收看六类测试。

>

> 有同学会问，讲本地化测试为什么要花一整节讲编码？因为编码错了，后面测的全是假的。你测出来的"翻译不准"，可能只是显示错了。测试人员至少要能判断这是编码问题、字体问题还是翻译问题，否则提的缺陷单会被开发退回来。能定位到层次，才提得出有效的缺陷。

提问：为什么讲本地化测试，要花一整节去讲字符编码？

预设回答：

- 答"因为乱码最常见"——对，这是最直接的理由。

- 答"编码不对，后面的测试全是假的"——更深一层：编码错了，你测的"翻译不准"可能只是显示错了。

- 答"编码是开发的事，测试不用管"——错。判断不了层次，缺陷单会被退回来。

- 收口：能定位到层次，才提得出有效的缺陷。

易错点 / 考点：① 五小节的顺序与关键词；② 8.1.2 和 8.3.2 的关系，前者讲原理，后者讲乱码的现象和检查点；③ 8.1.4 的八步是生产过程，8.4 的功能测试是验证活动，两件事别混。

过渡：进 8.1.1，先把 i18n 和 l10n 彻底分开。

【8.7】I18N vs. L10N（3 分钟）

> i18n、l10n 这两个怪词先说来历。internationalization，首字母 i 和尾字母 n 中间隔 18 个字母，缩成 i18n。localization，l 和 n 中间隔 10 个字母，缩成 l10n。再加一个 globalization，缩成 g11n，把前两个都包进去，指全球化这件事整体。

>

> 第一句，I18N 是 L10N 的基础和前提，为 L10N 做准备。这句定的是先后关系。没有国际化能力的软件去做本地化，是硬改，成本极高。像盖楼时没预埋水电管道，装修只能砸墙。

>

> 第二句，L10N 是 I18N 向特定本地语言环境的转换。留意"特定"两个字。国际化做一次，是通用的；本地化一个地区一份，简体一份、繁体一份、德语一份、阿拉伯语一份，每份都是独立的一次转换。

>

> 第三句，I18N 是软件产品源语言开发的一部分，属于 Engineering。这句决定责任归属。国际化是研发的活，写代码时就要做，不是测试阶段补的。测试能发现"这个字符串写死了"，改还得研发改。

>

> 第四句，L10N 可以独立于 Engineering，可由第三方完成。所以行业里有专门的本地化公司：资源文件发出去，翻译、回填、出多语言包，独立发包、独立验收。

>

> 课件底下还列着几个词：翻译、国际化、本地化、全球化。它们是一层层套起来的。翻译只是本地化里的一项，本地化被全球化包着，全球化管的是进哪几个市场、按什么顺序进。把翻译和本地化当成同义词，这一段的分就没了。

>

> 四句合成一张分工图：研发负责 i18n，本地化团队负责 l10n，测试两头都验——验研发留没留好接口，验本地化填没填对内容。后面所有内容都按这个分。

>

> 落到成本上一句话：i18n 做得好，l10n 是复制；i18n 做得差，l10n 是重做。

> **提问：**一个软件先做了德语版，再想做日语版，还要重新做国际化吗？

> **预设回答：**

- 答"要，每种语言都得重新国际化"——错。国际化是通用能力，做一次就够。

- 答"不用，再做一次本地化"——对。前提是当初做得干净：字符串抽出去了，格式没硬编码。

- 答"得看第一版怎么做的"——最专业。德语版要是硬改出来的，日语版就得再硬改一遍。

- 收口：这就是"技术债要不要还"的典型场面。

> **易错点 / 考点：**① 四句判断必背，特别是"i18n 属 Engineering、l10n 可第三方"；② g11n 不是第三种技术，是前两个的合集；③ 简答"国际化与本地化的区别与联系"，结构就是前提与转换、责任归属、复用性。

> **过渡：**既然 i18n 是前提，那"具备国际化能力"具体指哪些？课件给了八条。

【8.8】I18N（3 分钟）

> 国际化不是一句口号。课件给了八条能力，我按数据、代码、界面、语言习惯分四组，这样好记。

>

> 数据层两条。支持 Unicode 字符集、支持双字节字符。字符集不支持，后面所有翻译都是乱码。另一条是支持各国度量衡、时区、货币单位格式的设置。关键词是"设置"，意思是这些别写死，要能配。

>

> 代码层三条。分离程序代码和显示内容，消除 Hard code。这两条是一件事的两面：要给用户看的文字别写死在代码里，放进资源文件。翻译的人别去改源代码，一改就要重编、重测，一轮下来成本翻几倍。还有一条，用 Header files 定义经常被调用的代码段，把常量集中管，一处改动全局生效，不会漏改。

>

> 界面层两条。改善翻译文本尺寸，具有调整的灵活性。这条最容易被忽略：译文长度和原文不一样，按钮要能变宽，标签要能换行，对话框要能长大。界面按英文字符数精确排版写死的，德语一上就崩。另一条是国际化用户界面设计，界面本身要考虑不同文化的阅读习惯和审美。

>

> 语言习惯层两条。支持各国键盘设置，键盘布局不同，快捷键能用哪些字符就不同。支持文字排序和大小写转换，字母表顺序不同，排序规则也不同，有的语言压根没有大小写。这两条在中文环境下你感知不到，所以最容易漏。

>

> 八条收一句：把跟地区有关的一切从代码里赶出去，变成可配置、可替换的东西。

>

> 检查方法补一句。代码里没有中文，不等于没有硬编码。yyyy-MM-dd、$、kg、12 还是 24 小时，这些全是。查硬编码要搜格式串和单位符号，不只是搜中文。

> **提问：**八条里，哪一条最容易被团队"以为自己做到了"？

> **预设回答：**

- 答"消除硬编码"——很可能。界面上看得见的字符串容易抽出去，看不见的才要命：日志提示、异常信息、导出的文件名、报表表头。

- 答"文本尺寸可伸缩"——也很典型。源语言下每个按钮都刚刚好，一翻译全挤一起。

- 答"我们代码里没有中文，都做了"——这个回答本身就是陷阱。格式串和单位符号照样写死。

- 收口：所以查硬编码不能只搜中文，要搜格式串和单位符号。

> **易错点 / 考点：**① 八条要能归组记忆（数据、代码、界面、语言习惯）；② 高频错点是把"消除硬编码"只理解成界面文字，漏掉日志、异常、文件名、格式串；③ 简答问"国际化包含哪些方面"，答这八条比答"翻译"得分高得多。

> **过渡：**清单是抽象的，看一张真体现国际化能力的实例。

【8.9】国际化功能实例（2 分钟）

> 一个软件到底支不支持多语言，别看宣传页，看它切换时的反应。

>

> 第一类现象，同一个数字，显示的样子随区域变。一种设置下是 1,234.56，换一种变成 1.234,56，千分位

> 和小数点的符号对调。这说明程序没有自己拼字符串，是调了系统的区域格式。这就是"支持货币单位格式设置"落地的样子。

>

> 第二类，日期时间随区域变。24 小时制和 12 小时制，星期几怎么写，月份叫什么。

>

> 第三类，排序结果随区域变。同一组人名，两种语言下先后顺序不一样，因为字母表顺序不同。顺序还牵着检索和分页，顺序一变，翻页看到的内容也跟着变。

>

> 第四类，界面上冒出一个方框或者一串怪字符。这叫缺字形，属于字体或字符集的问题，跟翻译没关系。缺字形和乱码要分开，前者是字体里没这个字，后者是字节被解释错了。

>

> 这四类怎么用？把它当成一张验收照片。将来你测一个号称支持多语言的软件，就按四类各挑一个点，切区域设置，看它变不变。变了，能力在；不变，格式被写死了。

>

> 手法只有一个词：切换。切区域、切语言、切时区、切键盘，然后看同一个功能跟不跟着变。不切，你测的只是源语言那一种情况，等于没测国际化。切换顺序也有讲究：先切区域看格式，再切语言看译文，最后切时区看时间。一次只动一个变量，出了问题才知道是谁引起的。

>

> 提缺陷时三要素别漏：在什么区域设置下、期望什么格式、实际什么格式。缺一个，开发就得回来问你。

提问：把系统区域从中文切到德语，日期显示没变，这算缺陷吗？

> **预设回答：**

- 答"算，格式写死了"——大概率算，但先排除一种情况：这个字段可能有业务要求固定格式，比如对外接口报文。

- 答"不算，可能是缓存"——有可能，重启或清缓存再验一次，这是应有的严谨。

- 答"日期格式无所谓"——错。跨地区时日期格式会直接导致误读，2/3/2009 那个例子已经说明后果。

- 收口：缺陷单写清三要素——什么区域、期望什么、实际什么。

> **易错点 / 考点：** ① "切换"是核心手法，要能说出切哪四个维度；② 判断"显示不变"是不是缺陷，先确认有没有固定格式的业务要求；③ 缺字形（方框）和乱码是两种现象，考试常混。

> **过渡：** i18n 的能力说完，看另一半——l10n 到底要做哪些事。

【8.10】L10N（3 分钟）

> 翻译完了，本地化就做完了？差得远。课件列了六项，这个数字本身就是考点。

>

> 第一项，翻译。主体工作量在这儿，界面文字、提示、错误信息、帮助文档，都要术语一致、语气统一。

>

> 第二项，地区文化、宗教。图标、颜色、手势、动物形象、节日、地图边界，在不同文化里含义可能相反。某个国家视为禁忌的内容你不改，产品上不了架，甚至引发公关危机。这一项最容易被工程团队忽略。

>

> 第三项，度量衡和时区。公里还是英里，摄氏度还是华氏度，千克还是磅。时区不只是显示偏移，还有夏令时：有的国家有、有的没有，切换那一周最容易出日期错乱。

>

> 第四项，软件用户界面。翻完要重新排版：控件变宽、对话框变高、菜单换行、图标里的文字要重画。

>

> 第五项，联机文档，包括帮助文档和功能性 PDF。为什么强调功能性 PDF？有些软件的交付物本身就是 PDF：合同模板、报表模板、说明书。这些文件里的字体、编码、排版要单独处理，还常常不在研发的测试范围里，最容易漏。

>

> 第六项，热键设置。菜单项的快捷键跟字母绑，翻译之后字母变了，快捷键要重新分配，还要避开系统保留键。德语键盘上 Z 和 Y 的位置都跟英文不一样。

>

> 六项按工作量排，翻译最重，热键最轻。按风险排，文化和文档最容易被漏。最容易卡进度的是第二项和第五项，这两项常常不归研发管，出了事谁都不认。

>

> 记法：翻译、文化、度量、界面、文档、热键。你按自己的方式串，关键是别只记翻译。本地化的验收面，比"界面翻译"宽得多。

提问：六项里，哪一项最可能"研发完全不知道要做"？

预设回答：

- 答"文化宗教"——很对，这属于产品与市场范畴，研发通常不碰。

- 答"联机文档里的 PDF"——也对，PDF 往往由文档团队产出，测试不在链条上。

- 答"热键不用改，Ctrl+S 全世界一样"——常见错答。功能键通用，可菜单助记符是按语言分配的，翻完必须重排，否则两个菜单项会抢同一个字母。

- 收口：这六项的价值就是提醒你，验收面比"界面翻译"宽得多。

易错点 / 考点：① 六项必背，"只做翻译"是典型错答；② 热键问题的准确说法是菜单助记符要重新分配并避免冲突，不是功能键要改；③ "PDF 不属于本地化范围"——错，课件把它列进联机文档。

过渡：为什么编码问题总排第一？下到最底层看字符集。

【8.11】8.1.2 字符集问题（3 分钟）

> 乱码谁都碰到过。文件发过去，对面打开一片怪字符，你的第一反应常常是"文件坏了"。文件一个字节都没变。

>

> 课件给的定义很短：字符集是操作系统中所使用的字符映射表。映射表三个字是全页的关键，它说的是字符和数字之间的对应关系。计算机里没有"字"，只有数字。你看到的每个汉字，在机器里是一个编号。字符集就是这本编号簿。

>

> 课件列了五种：ASCII、GB2312、ISO8859-X、Unicode，加上 Unicode 的三种存法 UTF-8、UTF-16、UTF-32。它们不是并列的，是演进的。ASCII 只装英文，一个字节，128 个位置。GB2312 是中国自己的国标，为了装汉字，用两个字节。ISO8859-X 是欧洲的一堆单字节字符集，一个语系一份。Unicode 想一统天下，给全世界所有字符编号。UTF-8、UTF-16、UTF-32 是同一套编号的三种存法。

>

> 所以乱码的根因永远是一句话：写的时候用 A 号簿，读的时候用 B 号簿。一个 UTF-8 存的中文文件，被按 GBK 解码的程序打开，每个汉字被拆两半重新解释，出来的就是一串谁也不认识的字。反过来也一样。

>

> 还有个"半个字符"的问题。双字节字符占两个字节，程序截断字符串时按字节切，正好切在一个汉字中间，剩下半个字节谁也解释不了。所以"字段长度限制"在本地化测试里是高危点：英文字段限 20 个字符，中文也许只能放 10 个字，按字节算又会切坏。

>

> 还有一类表现更绕：数据库里存进去是对的，查出来是错的。连接串上没写字符集，客户端按本地代码页解释一遍，数据没坏，显示坏了。查这种问题，先看数据库那一层的字符集设置。

>

> 排乱码的顺序记住：先确认编码，再看字体，最后看翻译。顺序反了，白折腾。

提问：同一段中文，为什么在有的电脑上正常，在有的电脑上是乱码？

预设回答：

- 答"两台机器系统语言不一样"——方向对，更准确的说法是默认代码页或者解码方式不一样。

- 答"文件坏了"——错。文件一个字节没变，变的是怎么解释这些字节。

- 答"装个字体就好"——最典型的错答。字体只解决有没有这个字形，解决不了这段字节代表哪个字。

- 收口：排乱码的顺序是编码、字体、翻译，反了就是白折腾。

易错点 / 考点：① 字符集的定义与五种类型的演进关系；② 乱码和缺字形（方框、豆腐块）必须分清，前者是解码错，后者是字体缺；③ 高频考点"为什么双字节字符会导致截断出错"——按字节切可能切在字符中间。

> **【思政落点 1】**：字符集这一块最有意思的地方是，它出错的时候程序一切正常。没有异常，没有崩溃，日志也不报错，只是用户看到一片乱码。这类问题考的是细节上的较真：一个编码参数、一次按字节截断，看着都不影响运行，却直接决定用户能不能看懂界面。差距往往不在会不会写复杂算法，而在有没有人愿意为一个"看起来没事"的字节停下来查清。越是没人盯着的地方，越要自己盯。

> **过渡：**知道有哪几种字符集，再看国际上用什么标准把它们管起来。

【8.12】字符集问题标准（3 分钟）

> 上一节讲编码，这一节讲标准编号。五个编号看着像要背，每个都在回答一个具体问题。你把"它管什么"答上来，编号眼熟就行。

>

> 第一个，ISO/IEC 10646-1:2003。它定义四字节编码的通用字符集，缩写 UCS，也叫通用多八位编码字符集。它回答的是：全世界所有字符，怎么放进同一套编号体系。四个字节，空间够大，这也是 Unicode 的编号基础。

>

> 第二个，ISO 639-1:2002，两字母语种代码标准。中文 zh，英文 en，日文 ja，德语 de。它管的是"这是哪种语言"。

>

> 第三个，ISO 3166-1:1997，国家代码标准。中国 CN，美国 US。它管的是"这是哪个地区"。

>

> 语言和地区为什么要分开？因为同一种语言在不同地区规则不同。中文在简体环境叫软件，在繁体环境叫軟體，日期习惯、货币、度量也可能不一样。所以本地化标识通常是语言加地区，两段拼起来才够精确。拼出来长什么样，你天天在用：zh-CN、en-US、de-DE，前半是语言，后半是地区。写用例时把这个标识列出来，一个语言一个地区一行，覆盖面就清楚了。

>

> 第四个，RFC 3066，语言鉴定标签标准。它管的是标签怎么写才合法，也就是把前两个拼起来的语法规则。

> 标签写规范了，程序和翻译团队才有共同的说法。

>

> 第五个，RFC 3629，UTF-8。它管的是 Unicode 怎么用变长字节存下来。今天网页和接口默认用它。

>

> 五个词收一下：编号、语言、地区、标签、编码。考试口径也给一句：测试人员用得上的只有两个，知道语言代码和地区代码怎么拼出本地化目标，知道 UTF-8 有正式规范可查。剩下三个说出它管什么就够了。

> **提问：**本地化目标为什么不能只写"中文"，要写成"中文简体"或"中文繁体"？

> **预设回答：**

- 答"字不一样，一个简体一个繁体"——对，最直接的原因。

- 答"用词也不一样，软件和软体"——补得好，这是术语层面的差异。

- 答"中文就是中文，一样"——错。只写中文，翻译团队没法判断用哪套字形、哪套术语、哪套日期习惯。

- 收口：英语也一样，colour 还是 color，日期顺序也不同。最小单位是语言加地区。

> **易错点 / 考点：**① 五个标准各自管什么要能对上，别把 639（语言）和 3166（地区）搞反；② 课件给的编号与版本年份照抄，别自己改；③ 高频简答"为什么本地化标识要包含语言和地区两部分"。

> **过渡：**标准讲完，从最小的字符集看起——ASCII，所有编码故事的起点。

【8.13】示例：ASCII（2 分钟）

> 一张码表，看着枯燥，可它是整个编码世界的起点。

>

> 先看结构。ASCII 一共 128 个码位，7 个二进制位就够表示完，所以早期一个字节 8 位里，最高位是空着的。这一位空着，后面引出一堆故事。

>

> 排列有规律，记规律比背表有用。最前面一批是控制字符，编号最小，它们不显示成字，控制的是行为：回车、换行、制表、响铃。程序里写的 \n、\t 就是它们。

>

> 中间是数字，0 到 9 连续排列。所以"字符 7 减字符 0 等于数字 7"这种写法才成立。

>

> 再往后是大写字母，几个标点，然后小写字母。大小写的编号差固定是 32，英文里大小写转换才能靠加减一个常数做完。

>

> 最后是英文标点。

>

> 控制字符里有一对值得单独记：回车和换行。Windows 里一行结束是两个字符，Unix 里是一个。同一份文本在两边打开，行数可能对不上。这不是字符集问题，是平台约定，本地化测试里照样要查。

>

> 看这张表要看出局限：128 个位置，装得下英文、数字、英文标点，装不下任何一个汉字。中文常用字几万个，一个字节不够，必须两个字节。这就是 GB2312 出现的理由，也是"双字节字符"这个词的来历。

>

> 还有一条隐含信息：ASCII 里没有欧元符号、没有重音字母、没有阿拉伯文。欧洲人要写自己的语言，就得扩展它，这就是 ISO8859-X 存在的原因。

>

> 演进链也在这儿：一个字节不够，加字节；各国自己加；最后统一到 Unicode。

>

> 当年为什么不用空着的那一位多装字符？一种真实做法是用它做奇偶校验，早期通信确实这么干，不能随便拿来装字。另一层原因是当时 128 个够用。而这个"够用"，正是后来所有兼容问题的源头。

提问：ASCII 只用 7 位，一个字节还剩 1 位，当年为什么不拿它多装些字符？

预设回答：

- 答"当时没考虑别的语言"——对，标准是按英语世界的需求定的。

- 答"剩下那位用来做校验"——真实存在的一种做法，早期通信确实用最高位做奇偶校验。

- 答"128 个够用了"——当时够用，正是这个"够用"给后来埋了坑。

- 收口：标准的边界，往往就是后来所有兼容问题的源头。

> **易错点 / 考点：**① ASCII 是 7 位、128 个码位，别写成 8 位 256 个；② 大小写差 32、数字字符连续，这两个规律常出小题；③ "ASCII 可以表示中文"——错，这是双字节讨论的起点。

> **过渡：**一个字节不够用，就有人想一次把全世界装进去——这就是 Unicode。

【8.14】示例：Unicode（3 分钟）

> Unicode 干的事，是给每个字符发一个编号。课件给了三条信息，每条都关键。

>

> 第一，UCS-4，范围 0x0 到 0x10FFFF。意思是编号空间从 0 一直到十六进制的 10FFFF。四个字节理论上能装到 FFFFFFFF，没必要用那么多，划到这个上限足够覆盖现有和可预见的字符。

>

> 第二，1st 字节 0-16 个平面，后两个字节 0-FF 行、0-FF 列。这是个坐标式的读法：整个编号空间分成若干平面，每个平面里若干行，每行里若干列。一个字符的位置就是第几平面、第几行、第几列。听着像地图坐标，说的是编号分层。

>

> 平面数也常被考。平面编号从 0 到 16，第 0 个是基本平面，后面几个是辅助平面。数的时候别从 1 数起，第 0 个也是一个平面。

>

> 第三，0 平面叫 BMP，Basic Multilingual Plane，也就是 UCS-2。这是全页最重要的一句。几乎日常用的字符都在这个平面：拉丁字母、汉字、日文假名、阿拉伯文、标点。所谓 UCS-2，就是用固定两个字节表示这一层的字符。前面说的双字节字符，指的就是这一层。

>

> 由此能推一个实用结论：绝大多数情况下，一个汉字在 UCS-2 里占两个字节，在 UTF-8 里占三个字节。同一句话，不同编码下字节数完全不同。这正是"按字节限制长度"会出事的原因。

>

> 再补一句，基本平面之外的字符，比如生僻字、古文字、表情符号，要用代理对表示，两个双字节拼起来。这类字符是本地化测试的高危区：长度算不对、截断截坏、排序乱掉，全从这儿来。

>

> 所以碰到"长度限制"，第一件事问清限的是字节还是字符，第二件事拿最长的目标语言去试。

> **提问：**一个字段限长 10 个字节，能存几个汉字?

> **预设回答：**

- 答"10 个汉字"——错，把字节当成字符了，这是最普遍的错误。

- 答"5 个"——按 UCS-2 每字两字节算是 5 个；内部要是 UTF-8，汉字占三个字节，只能放 3 个。

- 答"要看编码方式"——最准确。先问编码，再算字数。

- 收口：问清限的是字节还是字符，再拿最长的目标语言去试。

> **易错点 / 考点：**① BMP、UCS-2、双字节三者的关系要串起来讲；② "一个汉字就是一个字节"——错，这是本地化缺陷的高发根源；③ 编号范围 0x0–0x10FFFF 常作填空，平面编号是 0 到 16。

> **过渡：**Unicode 解决了编号，没解决怎么存，这就是 UTF-8 要回答的。

【8.15】示例：UTF-8（2 分钟）

> 同一句话，三种存法，字节数不一样。UTF-8 是今天互联网上最常用的那一种。

>

> 它的核心思想是变长：一个字符用几个字节，看它是哪个字符。英文 26 个字母还是 1 个字节，跟 ASCII 完全一样。带重音的欧洲字符 2 个字节。常用汉字 3 个字节。更生僻的、表情符号 4 个字节。

>

> 变长带三个后果，每个后果都是一类测试点。

>

> 第一个，向后兼容。纯英文文本，用 UTF-8 存和用 ASCII 存，字节完全一样。这就是它能迅速取代各种本地编码的原因：老系统不用改就能读。

>

> 第二个，长度不再等于字符数。这条最要命。程序里所有"取前 N 个字符""截断到 N 字节""按长度校验"的地方，在 UTF-8 下都得小心。截断切在一个三字节汉字的中间，剩下两个字节就是非法序列，显示出来是乱码或者问号。

>

> 第三个，解码可以自同步。UTF-8 让每个字节的高位能表明自己是单字节字符，还是多字节字符的第几个字节。好处是数据流中间丢了一段，程序还能从下一个合法起点重新对齐，不会一路错到底。这是它比很多老编码健壮的地方。

>

> 还有一处要提：UTF-16 得管字节序，同一段文字有大端小端两种存法，文件头还得带个标记。UTF-8 没有这个问题，跨系统传文件更省事。

>

> 课件给的链接是 RFC 3629，UTF-8 的规范原文。课上不要求你读，但要知道一件事：编码有正式规范，不是各家自己商量。遇到编码争议，最后要回到规范上说话。有标准的地方，别靠猜。

>

> 还有一类现场表现要会判：UTF-8 存的文件被只认 ASCII 的老程序读，英文毫发无损，中文全乱。这个"看着打开了、内容却不对"的状态，最危险。

提问：一个用 UTF-8 存的文件，被只认 ASCII 的老程序读，会发生什么？

预设回答：

- 答"中文变乱码"——对，非 ASCII 部分全部读不出来。

- 答"英文正常，中文乱"——更准确。ASCII 段兼容，所以英文毫发无损。

- 答"文件打不开"——不一定，多数是能打开但内容错，这种"看着打开了、内容却不对"的最危险。

- 收口：所以本地化测试要专门留一条"在非 Unicode 环境下的表现"。

> **易错点 / 考点：** ① UTF-8 是变长编码，1 到 4 个字节；② "长度不等于字符数"是本地化缺陷的高发点，要能举例；③ "UTF-8 是 Unicode 的另一种字符集"——不准确，Unicode 是字符集和编号，UTF-8 是它的编码方式之一。

> **过渡：** 底层讲完，把视角抬到设计层——做到哪五条，才算具备国际化标准。

【8.16】8.1.3 软件国际化标准（3 分钟）

> 设计评审时拿什么去量一个软件的国际化够不够格？课件给了五条。这五条和前面那八条角度不同：八条回答要具备什么能力，五条回答接口做成什么样才算合格。每条我都带一个测试点。

>

> 第一条，切换语言的机制。软件要能换语言，换完别要求重装、重启。测试点：切换入口在哪，切完界面有没有立刻变，有没有一部分文字没跟着变。没跟着变的，通常是缓存或者写死的地方。

>

> 第二条，与语言无关的输出接口。程序往外吐的东西——日志、导出文件、报表、接口报文——格式不能依赖当前语言。一旦依赖，同一个系统在不同语言环境下产出的数据就不一致，下游解析不了。测试点：同一份数据在两种语言环境下导出，要一样的必须一样。

>

> 第三条，与语言无关的输入接口和标准的输入协议。这条和第二条对称。输入不能只认一种编码、一种格式。用户可能在中文输入法下敲全角符号，可能粘一段 UTF-8 文本。测试点：全角半角混输、粘贴多语言文本、导入外来文件。

>

> 第四条，资源文件的国际化。要翻译的内容放进资源文件，还要有管理机制：分文件、分模块、能增量更新、缺项能回退。

>

> 第五条，支持和包容本地化数据格式。数字、货币、日期、时间、度量衡，都能按目标地区的习惯来。

>

> 五条收成五句：界面可切、输出中立、输入宽容、资源可换、格式可配。评审一份国际化设计方案，就拿这五句一条一条问能不能做到。答不上来的那条，就是后面要补测试的地方。

提问：第四条里，某个词条忘了翻译，你希望界面显示什么？

预设回答：

- 答"显示英文原文"——常见的降级策略，用户好歹能猜。

- 答"显示空白"——最差的选择，用户既不知道是什么，也不知道出了错。

- 答"显示 %s 或者 BTN_OK 这种键名"——用户看不懂，可它一眼就能被看见，比静默回退更容易发现漏译。

- 收口：验收前把策略定死：漏译回退到源语言，同时让测试能定位到是哪一条漏了。

易错点 / 考点：① 五条要求能默写关键词；② 输入接口和输出接口是成对的两条，别只写一半；③ "国际化只是界面的事"——错，第二、三条明确要求输入输出与语言无关。

过渡：设计标准有了，再看测试怎么切分——i18n 测试和本地化测试查的不是同一批东西。

【8.17】I18N 测试（3 分钟）

十四个问号排成两列，左边国际化支持，右边本地化测试。带走的是分工：左列查能力，右列查填充。

>

> 左列七个：MBC 字符集和脚本支持吗？MBC 的输入和显示正常吗？MBC 的文件名、文件夹名，处理对吗？区域设置生效吗？索引和排序正确吗？本地化操作系统上能跑吗？键盘支持吗？

>

> 这七个的共同点是，在源语言版本上就能测，不必等翻译交付。文件名支持多字节，你拿一个中文文件名在英文系统上建一下就知道了；区域设置生效，切一下看日期变不变就知道。这是 i18n 测试最大的优势：早，而且便宜。

>

> 右列七个：本地化资源齐全吗？区域设置有没有被硬编码？链接和串联的字符串有没有问题？资源文件里有

> 没有夹着非本地化内容？文字有没有扩展余地？非层次图像里的文字处理了吗？别的组件呢？

>

> 右列的共同点是，要等翻译交付之后才能测，缺陷也更贵——改一处要重译、重排版、重回归。

>

> 课件还标了一条：Pseudo-translation，伪翻译，重要部分。翻译交付之前，先把资源文件里的英文机械换成一段变长的、带特殊字符的假文本，再跑一遍界面。几分钟就能暴露三类问题：文字扩展装不下、硬编码没抽出来、编码处理有问题。成本几乎为零。

>

> 最后给个术语：MBC，Multiple Byte Character，多字节字符集。看到它就想中文、日文、韩文这类占多个字节的字符。

>

> 单拎一项说清，"链接/串联字符串"为什么值得单独测。程序里常写"共 " + 数量 + " 条记录"，译成别的语言语序可能整个反过来，拼出来就是病句。正确做法是用带占位符的完整句子让译者看到整句，测试专门去找界面上那些由几段拼出来的话。

> **提问：**为什么"链接/串联字符串"会成为一个单独的测试项？

> **预设回答：**

- 答"句子拼起来语序会乱"——对。程序里常拼"共 N 条"，换个语言语序整个反过来，拼出来是病句。

- 答"翻译只翻了片段，前后接不上"——也对。句子被切段分译，译者看不到上下文，术语和语气就不统一。

- 答"拼接没什么问题，能显示就行"——错。这是本地化最典型的隐性缺陷，源语言读着通顺，换语言就成了机器翻译腔。

- 收口：用带占位符的完整句子，让译者看到整句；测试专门去找拼接出来的话。

> **易错点 / 考点：**① 两列各七个问号说出大意即可，重点是分清哪一列能在源语言版上测；② 伪翻译是本章高频名词，要能说出作用和成本优势；③ MBC 的英文全称常考填空。

> **过渡：**左列能提前测，具体用什么方法？下面给三种。

【8.18】国际化测试方法（2 分钟）

知道要查什么，还得知道怎么查。三种方法，每种对应一个明确场景。

>

> 第一种，设计评审和代码审查。静态方法，不运行程序。评设计里有没有把格式、语言、地区写死；审代码里有没有硬编码字符串、有没有按字节截断、有没有用系统默认编码读写文件。为什么排第一？这类问题越早发现越便宜。等到测试阶段才发现"整个模块的日期都是拼字符串拼出来的"，改造成本可能是重写。

>

> 第二种，针对源语言的功能测试，比如不同的区域设置、不同的时区显示。这里的关键是用词——界面还是英文或中文，但我把系统区域切到德国，把时区切到纽约，看功能还对不对。这一类最划算，不需要任何翻译交付，却能暴露大量区域相关缺陷。时区特别值得测：跨时区、跨夏令时切换点、跨日期边界，最容易出"差一天"的毛病。

>

> 第三种，针对伪翻译版本的测试。资源文件里的文本换成变长的假文本，跑起来看界面。它专治三类病：布局被撑破、文字被截断，以及那些没抽进资源文件、所以没被伪翻译覆盖到的硬编码。最后一类最值钱：凡是没有变成假文本的地方，就是硬编码的地方。这一招叫反向定位。

>

> 三种的关系：评审管设计上对不对，源语言测试管区域相关功能对不对，伪翻译测试管界面和资源处理对不对。三层做完，国际化这块基本就干净了。

>

> 有同学会问，伪翻译这么便宜，能不能替掉评审？替不掉。伪翻译只在界面上说话，代码里按字节截断、用默认编码读写文件，它一个字都不会提。

提问：伪翻译测试为什么能"反向定位"硬编码？

预设回答：

- 答"假文本很显眼，一眼看出哪些没被替换"——对，这就是核心逻辑。界面上还留着正常英文的地方，就是没走资源文件的。

- 答"它会把文字变长，撑破的地方就是布局问题"——这是它的第二个作用，不是反向定位。

- 答"伪翻译就是随便乱码一下"——错。伪翻译是有规则地替换：变长、加特殊字符、加边界标记，目的性很强。

- 收口：它既是压布局的手段，也是找漏抽字符串的探针。

> **易错点 / 考点：** ① 三种方法按静态、动态分类；② 伪翻译的两大作用（压布局、找硬编码）是简答得分点；③ "国际化测试必须等翻译完成才能做"——错，前两种都不用翻译。

> **过渡：** 方法有了，具体查哪些点？下面是一张十项清单。

【8.19】国际化测试点（3 分钟）

> 十项清单，最后还有个省略号，意思是还能往下加。我按输入、显示、转换、交互四组归，每项说一句怎么测。

>

> 输入组两项。多字节和单字节文字混合：一句话里同时敲中文、英文、数字、全角符号，看显示和存储。输入法编辑器 IME：候选框、联想、上屏，在某些界面会出事，比如输入框被候选框挡住、上屏后光标跳错位。

>

> 显示组两项。双向识别功能，就是刚才的 RTL，一段文字里混了两个方向，程序要能判出每一段的走向。大小写转换和换行：有些语言没有大小写，转换函数遇到它们会不会出错；换行规则也不同，中文能在任意字之间断，英文必须按词断，日文还有自己的一套禁则。

>

> 转换组两项。硬编码，找没抽进资源文件的文字和格式。语言切换方式，切换入口、切完是否立即生效、有没有残留。

>

> 交互组四项。快捷组合键，翻译后菜单助记符要重新分配，别让两个菜单项抢同一个字母。纸张大小，A4 和 Letter 尺寸不同，打印排版会错位，打印报表、发票、合同的软件必测。电话号码，位数、分组、区号、国际前缀各国都不同，输入校验按中国手机号写死 11 位，出国就废。词序问题，主谓宾的顺序、形容词和名词的先后、数量和单位的组合，各国不同，靠拼字符串组句子一定出问题。

>

> 抓手给你一个：混、法、切、硬、换、键、纸、电、序，再加一个双向识别。十项抄在测试计划第一页，做国际化测试时逐项打勾，别凭印象。

提问：电话号码怎么也进了国际化测试点？它不就是一串数字？

预设回答：

- 答"各国位数不一样"——对，中国手机 11 位，美国 10 位，有些国家还带前导零。

- 答"格式不一样，有的加括号有的加横线"——也对，显示格式和校验规则都要跟着地区走。

- 答"电话号码是业务字段，跟本地化没关系"——错。凡是有固定长度、固定格式、固定校验规则的字段都是高危点，邮政编码、身份证号、税号、车牌同一类。

- 收口：判断一个字段要不要本地化，就问一句"这个规则在别的国家还成立吗"。

易错点 / 考点：① 十项要能归类记忆，常以多选题出现；② 纸张大小最容易被忘，它属于打印排版类；③ 高频案例"输入框写死手机号 11 位"是本地化缺陷的经典题。

过渡：清单讲完，看几组截图，把这些抽象的点落到眼睛里。

【8.20】示例（2 分钟）

最后落到截图上。四类判断，一个个过。

>

> 第一类，看文字装不装得下。有文字被截断、有省略号、有半个字露在外面，这是布局问题。判断要点是分清控件太小还是文本太长：控件按源语言长度写死的，那是设计问题，不是翻译问题。

>

> 第二类，看字符显不显示得出来。有方框、有问号、有一串怪字符，是编码或字体问题。这里要分开：方框通常是字体缺字形，成串的怪字符通常是解码用错了字符集。两者修法完全不同。

>

> 第三类，看方向和对齐。RTL 里要右对齐的地方左对齐了、镜像没做、括号反了，都在这儿。

>

> 第四类，看语言混排。中英文数字混在一起，间距、基线、换行位置正不正常。这条最考验实现，也最难自动检测。

>

> 四类之外还有一类不好归：文字长短变了，列宽没变，表格里的数字开始折行。看着不像错，读起来费劲，可以算到布局里。

>

> 还有一个职业习惯要说：看截图不能只说"哪里不对"，要能说出为什么不对、要改成什么。缺陷单写"界面显示异常"，开发没法处理。写"德语环境下保存按钮的文字超出边框约两个字符宽度，建议按钮宽度自适应或允许换行"，开发一看就知道怎么动。

>

> 判断是译文太长还是控件太小，最快的办法是切回源语言看。源语言正常，就是译文长；源语言也挤，就是控件写死了。更根本的判断是看控件有没有固定宽度。所以好缺陷单会写"建议控件宽度自适应，而不是缩短译文"。

> **提问：**截图里一个按钮上的字被切掉了，你怎么判断是翻译太长还是控件太小？

> **预设回答：**

- 答"切回源语言看，正常就是译文太长"——最快的判断法，很实用。

- 答"看控件有没有固定宽度"——从实现角度看，更根本。

- 答"反正是显示问题，让开发自己看"——错。不区分原因，开发可能只缩短译文，写死宽度的地方还在，换个更长的语言照样犯。

- 收口：好的缺陷单会写"建议控件宽度自适应，而不是缩短译文"。

> **易错点 / 考点：**① 四类问题的分类判断是案例题基本功；② 缺陷描述的现象、环境、建议三要素，给一段糟糕描述让你改时按这三条补；③ 别把字体缺字形和编码错误混成一件事。

> **【思政落点 2】**：把缺陷写清楚，是件没人监督的事。"界面显示异常"六个字也能交差，写清环境、现象、建议要多花十分钟。区别不在能力，在愿不愿意替下一个人多想一步。测试报告既是给同事看的，也是给半年

后的自己看的：只凭这句话，回归的人能不能复现。职业素养大多落在这种没人检查的十分钟里。

过渡：从这里开始进流程——本地化这件事，从头到尾分八步。

【8.21】8.1.4 软件本地化基本步骤（3 分钟）

你们有没有算过，把一个已经上线的软件做成德语版，到底要动多少东西？多数人第一反应是：把词条导出来，翻完再导回去。真这么干，两周以后一定出事。

>

课件把这件事拆成八步，我按五段串。第一段是准备，两步。先建配置管理，把目标语言的每个版本跟源代码对应上。不对应会怎样？中文版在改、德语版也在改，你说不清德语版是从哪一版翻出来的，新功能上线就漏一段没译。第二步是术语表，而且要求维护。同一个词，界面叫导出，帮助文档叫输出，报表里叫下载，用户会当成三个功能。

>

第二段提取。源语言代码和资源文件分开，把要翻的文本抽出来。抽不干净，译者手里会混进变量名和日志，翻完还会被塞回界面。

>

第三段翻译。课件写的是文本、图片等。图片里有字的，往往得重画，不是翻。

>

第四段回填。翻好的东西重新检入目标语言的源代码版本。走配置管理，不靠手工粘贴。

>

第五段验证，三步。需要就编译。然后测翻译后的软件，按译文调界面，这一步算返工。最后完整测一遍，格式和内容都对。

>

八步走完你会发现，翻译只占一步，剩下七步全是工程和管理。所以本地化项目栽跟头，很少栽在译者水平上。

提问：术语表为什么要放在翻译之前，而且要求维护，不只是创造？

预设回答：

- 答"不统一，用户会以为是两个功能"——最直接的后果，也是最常见的投诉。

- 答"换人、换外包，得有依据"——很实际。人一流动，术语表是唯一能保证一致的东西。

- 答"术语表是翻译的事，测试不用管"——反过来。测试拿它当判据：译文里的词跟术语表不一致，就是缺陷。

- 收口：没有术语表，"翻得好不好"就只能靠感觉。

易错点 / 考点：八步的顺序和名称，特别是调界面在完整验证之前，别颠倒。高频判断"本地化的主要工作是翻译"——错，八步里只有一步是翻译。术语表的作用答一致性加可判定，简答常给分。

【思政落点 3】：八步里最不显眼的是第一步配置管理。它不产出任何看得见的东西，却决定后面七步有没有依据。本地化出的真事故，很多不是译错，是译文没错、对的是上一版界面。这种事靠聪明解决不了，只能靠留痕：谁改的、改在哪个分支、对的是哪版译文。流程不是给笨人准备的，是让普通人也能稳定交付。

过渡：八步是做事的顺序。放进产品生命周期里看，它站在哪一段？

【8.22】本地化过程（2 分钟）

排期表上，本地化常常被排在最后两周。功能做完，找几个译者，翻完发版。真按这个排，第一次出海就得返工。

>

课件把整条链子摆出来了。先是国际化软件设计。接着是核心功能测试、国际化测试、翻译测试、本地化测试，最后落到全球化软件发布。

>

两个位置值得多看两眼。第一个，国际化软件设计在最前面。设计没做，后面的测试只能把缺陷挑出来，不能把能力造出来。日期格式硬编码在代码里，你测一百遍它还是硬编码。

>

第二个，核心功能测试排在国际化测试之前。为什么？先得确认这软件在源语言下功能是好的。功能本身有

> 毛病，你去测国际化，测出来的失败你说不清是功能缺陷还是本地化缺陷。先钉住基线，再测增量，这是测试里最基本的纪律。

>

> 再往后，国际化测试查能力，翻译测试查译文本身：完整性、术语、语境。本地化测试把翻好的版本当完整软件测一遍，安装、升级、打印、帮助文档都在里面。

>

> 最后落到全球化软件发布。发布也是一个环节，不是终点：多语言版本要一起发，版本号要对齐，语言包要能单独更新。

>

> 还有一件事得提前说。这条链子会回流。翻译测试发现译文和界面对不上，退回回填；本地化测试发现控件装不下，退回调界面再重新构建。排期排成一条直线，返工就把进度吃掉了。

> **提问：**翻译测试为什么单独成一个环节，不并到本地化测试里？

> **预设回答：**

- 答"译文问题量大，要集中查"——有道理，问题确实集中在语言层面。

- 答"做的人不一样，翻译测试常由语言专家做"——准确，这是组织上的理由。

- 答"没必要分，一起测"——小项目可以合并。多语言并行一混，缺陷归属就说不清：是译文错，还是程序显示错。

- 收口：分开是为了归因干脆。语言问题给译者，显示问题给开发。

> **易错点 / 考点：**六个环节的顺序必背。核心功能测试排在本地化之前，这条纪律常考。高频判断"本地化测试可以替代翻译测试"——错，两者的判据和责任人不同。

> **过渡：**链子走完了，回头看 8.1 最后一小节：本地化测试到底要干哪六类活。

【8.23】8.1.5 软件本地化测试（3 分钟）

> 有人说过一句话：原版都测过了，本地化版还测什么。这句话听着省事，代价一般落在上线以后。

>

> 本地化版是重新编译出来的另一个产品。资源文件换了，代码有时候也动了，安装包不一样，升级路径不一样。原版通过，说明不了这一版通过。

>

> 课件列了六类。功能性测试，所有基本功能、安装、升级。六类里，安装和升级最容易被跳过，也最容易出事。语言包能不能装上？装错了语言能不能切回来？升级完译文会不会被覆盖成英文？

>

> 翻译测试，语言完整性和术语准确性。完整性看有没有漏译、有没有英文残留；准确性看术语统不统一、语气对不对、有没有机翻痕迹。

>

> 可用性测试，界面、度量衡、时区。度量衡和时区放在这里不是笔误。用户看到三磅还是一点四公斤，看到时间差八小时，都是好不好用的问题。

>

> 兼容性调试，硬件、版本。关键词是调试，这类问题往往要反复试：换系统、换语言包、换输入法、换打印机驱动。

>

> 文化、宗教、喜好这类适用性：地图边界、宗教符号、颜色寓意、手势、节日、历史表述。这条最软，出事最硬，弄错了不是缺陷是事故，可能直接下架。

>

> 手册验证，联机文件、在线帮助、PDF。截图是不是旧版本、链接能不能跳、索引能不能搜、字体嵌没嵌进去。

>

> 六个词：功能、翻译、可用、兼容、文化、手册。前四类是技术活，后两类考分寸。

> **提问：**安装和升级为什么必须重测？本地化版在安装上跟原版差在哪？

> **预设回答：**

- 答"安装包不一样，多了语言文件"——最直接的原因。

- 答"要测语言选择、语言切换"——也对。安装向导本身要翻，而且得在装的时候就能选语言。

- 答"安装程序没改，不用测"——错。向导文字要翻，安装路径可能有非 ASCII 字符，升级时旧配置还可能盖掉新译文。

- 收口：第一条用例常常是，在一台目标语言的干净机器上，从零装一遍，再升一遍。

> **易错点 / 考点：**六类内容不能漏，顺序不必死记。度量衡和时区归在可用性里，答题按课件口径写。高频判断"功能在原版测过，本地化版可以免测"——错，它是另一个构建产物。

> **过渡：**六类说的是测什么，下面看怎么分工。

【8.24】软件本地化测试（2 分钟）

一条缺陷提上去，被退回来三次，在本地化项目里不算稀奇。退回来的理由常常是：这块不归我改。

>

> 课件把测试切成六块，跟上页切法不一样，这一刀是按执行分工切的。I18n 测试查国际化能力，源语言版就能做。L10n 测试查本地化版本本身。语言和翻译的测试管纯语言层面：漏译、错译、术语、语气。美观和界面的测试管排版、对齐、截断、控件布局。功能测试看业务在本地化环境下还对不对。发布测试管打包、安装、升级、许可、版本号、语言包分发。

>

> 为什么把语言和美观分开，把功能和发布分开？因为责任人和修法不一样。语言问题返给译者，美观问题返给前端，功能问题返给开发，发布问题返给构建和运维。分对了，单子当天流转；分错了，三个人互相踢。

>

> 课件给了两个参考。一份是 Mozilla 的 Firefox 2.0 L10n 测试计划。另一份是 OpenOffice 的本地化测试计划。两份都是开源项目公开的文档。价值不在结论，在于你能看见工业级的计划长什么样。哪些测试项、多语言怎么组织、每种语言的状态怎么记，都摆在那儿。

>

> 课后真去翻一翻。翻的时候带三个问题：它列了哪些测试项？每种语言怎么标完成度？缺陷怎么归类？这三

> 问的答案，就是你写计划的模板。

>

> 这类计划里通常有一张语言状态表：未开始、翻译中、待验证、已通过。多语言并行的时候，这张表就是项目管理的抓手。没有它，进度只能凭感觉汇报。

> **提问：**德语读着别扭，和德语按钮文字被截断，这两条分到哪一块？

> **预设回答：**

- 答"都是翻译问题"——错。一个是语言，一个是布局，修的人和方法完全不同。

- 答"别扭给译者，截断给前端"——正确，这就是分类的意义。

- 答"都提给开发，让他判断"——可以，效率最低。

- 收口：六个名词不是重点，分类即分派才是。

> **易错点 / 考点：**这六块按测试类型和责任人分，上一页六类按测试内容分，别混。两个参考计划的出处要能说出：Mozilla、OpenOffice。案例题给一条缺陷让你归类，盯住语言和美观的界线。

> **过渡：**六块里前两块最容易混。i18n 测试和 l10n 测试，差在哪？

【8.25】I18N 测试 vs. L10N 测试（3 分钟）

> 同一个软件，两份报告摆在桌上。一份写着 i18n 全部通过，一份写着 l10n 不合格。学生常问：这不矛盾吗？

>

> 不矛盾，两个测试量的不是一件事。五个维度足够说清。

>

> 第一维，测什么。i18n 测能力，这软件有没有被本地化的资格。l10n 测成品，这个特定语言的版本做得好不好。

>

> 第二维，什么时候做。i18n 早，源语言版就能做，设计评审阶段就能开始。l10n 晚，得等译文交付、重新构建之后。

>

> 第三维，语言环境。i18n 通常在单语言环境下做，界面还是源语言，只切区域、时区、键盘。l10n 必须在目标语言环境里做。装目标语言的操作系统，装目标语言的输入法，有条件就到目标国家实测。

>

> 第四维，缺陷归谁。i18n 的缺陷基本都是开发的：硬编码、按字节截断、默认编码写死。l10n 的缺陷可能属于译者，可能属于界面，也可能属于开发。归属不同，返工量差一个数量级。

>

> 第五维，能不能复用。i18n 测一次，对所有语言都有效。l10n 每种语言来一遍，语言越多总量越大。

>

> 五维里最要紧的是第二维。越早发现越便宜，这条在软件工程里几乎找不到例外。所以业界的做法是把 i18n 做扎实，让 l10n 专心管语言和排版。一个是体检，一个是装修验收。

>

> 一句话记住：i18n 决定能不能，l10n 决定好不好。两者也替代不了对方。i18n 全过，l10n 照样可能因为译文差而不合格。反过来，l10n 全过，也不证明这软件撑得住第十种语言。

提问：i18n 测试全过，能不能说这软件本地化没问题?

> **预设回答：**

- 答"能，底子好"——错。底子好只说明能装，不说明装得对。

- 答"不能，还得看具体语言的译文和排版"——对，这是 l10n 的活。

- 答"还得看目标国家的文化禁忌"——补得好，属于 l10n 的适用性。

- 收口：i18n 管能不能，l10n 管好不好。

> **易错点 / 考点：**五维对照是简答的标准骨架。高频判断"i18n 通过即代表本地化质量合格"——错。单语言环境和目标语言环境这一维最容易漏，补上能加分。

> **过渡：**8.1 到这儿结束，进 8.2，翻译要验哪三样东西。

【8.26】8.2 翻译验证（2 分钟）

> 用户的抱怨往往不是你们翻错了，是这软件看着不像给我们的。差别就在翻译验证。

>

> 课件给了三项。第一项，翻译的内容。这是基本盘。漏译，源语言有、目标语言没有。错译，意思歪了。多译，源语言删了、译文还留着。术语不统一。占位符被译坏。占位符这一条最技术。句子里有个 %s 或者 {0}，译者顺手翻成中文，程序跑起来要么崩，要么错位。

>

> 第二项，目标语言的文化心理。这一项最微妙。同一句话语法全对，读起来不像人话。面向消费者的轻松语气译成公文腔，正式场合的敬语译成随口一说。还有敏感度：有些表达字面没错，在那个文化里有冒犯意味。这一类机器查不出来，只能靠目标语言的母语者读一遍。

>

> 第三项，特殊符号。中文引号和英文引号、破折号、省略号、货币符号、度量单位符号、数学符号、版权符号。这些符号在各语言里的形态和用法不一样，翻译时最容易被原样照抄，抄完很别扭，甚至显示不出来。

>

> 三项收到一句：翻译验证不是查错别字，是判断这段文字放在这个文化里，像不像本地人写的。

>

> 所以本地化行业认的黄金标准是母语者审校，不是拿双语词典逐词对照。词典能告诉你意思对不对，告诉不了你这句话念出来别不别扭。

> **提问：**译文语法全对、意思也对，用户说读着别扭，这算缺陷吗？

> **预设回答：**

- 答"算，可读性问题"——算。这类在本地化项目里很常见，一般定中低级，但必须改。

- 答"不算，意思对就行"——典型错答。软件给人用，读着别扭，用户会怀疑是不是山寨的。

- 答"看产品类型，正式文档要求高"——专业的补充，产品定位决定语言标准的严格程度。

- 收口：验收标准要提前写清，是能看懂，还是读起来自然，两档返工量差很多。

易错点 / 考点：三项必背。占位符被译坏是技术性最强的考点，要能举 %s、{0}。高频判断"翻译验证可以靠不懂目标语言的工具完成"——只能做一部分，语气和文化心理得母语者判。

过渡：翻译只是本地化的一半，另一半是翻译之外的改动。

【8.27】软件本地化与翻译（3 分钟）

出海栽的跟头，一多半不在译文上。译文能找人改，改不动的往往是那些没人想起来要动的东西。

>

本地化是两层活。技术层面五项。调整大小，控件、对话框、列宽要装得下译文。调整默认设置，默认语言、区域、纸张、货币按目标地区设。重新编译，本地化版是重新构建出来的产物。创建新的图形，图里带字的、图标带英文缩写的都要重做。重新编排文档格式，页面尺寸、字体、目录、索引全要重排。

>

这五项有个共同点：都能量化。控件宽度够不够，默认值对不对，编译报不报错。图里文字换没换，文档页码乱没乱。一条条都能写成用例。

>

文化层面不一样。课件列的是包装、图标、宣传、样品、政治敏感的术语、地方规章和宗教信仰。这一层没有对错，只有合适不合适。同一个手势，有的国家表示可以，有的国家是侮辱。同一个颜色，一处喜庆，一处办丧事。地图上的边界怎么画，在某些国家是法律问题。

>

两层为什么要分开？处理方式根本不同。技术层靠测试和返工，文化层靠评审和本地顾问。你让一个不了解当地文化的测试团队去判断图标合不合适，他判断不出来。不是他不专业，是他手里没有那份信息。

>

成熟的做法是两道关：技术层由测试兜住，文化层建一道本地审校加合规评审。两道都过，才允许发布。

提问：一个竖大拇指的图标，当作点赞放进产品，需要本地化评审吗？

预设回答：

- 答"需要，有些地区这手势是冒犯"——对，文化层面的典型例子。

- 答"不需要，这是通用图标"——最常见的错答。正是这种想当然的通用，造成大量出海事故。

- 答"看目标市场，只发国内就无所谓"——合理。文化评审的范围跟发布范围挂钩，这是成本意识。

- 收口：文化检查项按目标市场逐条过，不是全球一套标准。

> **易错点 / 考点：**两层各自的项要能分开举例，别把重新编译写进文化层。高频判断"文化适用性问题是缺陷"——更准确的说法是适用性、合规问题，处理路径和功能缺陷不同。简答问"本地化为什么不等同于翻译"，就用这两层答。

> **【思政落点 4】**：文化这一栏考的是尊重和合规。有些内容我们看着平常，在别人那里是禁区；有些表述国内没问题，到了别的国家碰上法律。这不是要不要迁就谁，是两条底线：尊重是跨文化合作的基本教养，合规是产品能不能在当地合法卖的前提。将来做海外项目，不确定的去问当地人，拿不准的按最严的标准处理。

> **过渡：**技术层面第一件要改的是大小，译文一长，界面就装不下。

【8.28】翻译问题 – 文字扩展（2 分钟）

> 德语版第一版发出去，最常见的抱怨是：按钮上的字没了。

>

> 原因不复杂。同一个意思，不同语言写出来长度差很多。课件有句原话：德语最长，汉语比较精炼。这不是印象，是结构差别。德语喜欢把词拼成很长的复合词，一个词能顶中文一句话。中文单字表意，同样的信息用更少的字符就装下了。反过来，中文译成英文多数也会变长，英文要多用词，还得加冠词介词。

>

> 长出来会打到哪儿？五个地方。按钮和标签，宽度按源语言定的，一长就截断，或者挤成两行，两行又把工具栏撑高。表格列，列宽固定，长文字要么截断，要么把整张表推到屏幕外。对话框和提示，文字超边界，按钮被挤出可视区，用户点不到确定。菜单，菜单项换行，子菜单跑到屏幕外。帮助文档和错误信息，段落长度变了，页码、索引、交叉引用全得重排。

>

> 测这一类，办法就一句：用最长的语言去试。业界的做法是先做伪翻译，把文本机械拉长。布局问题在译文

> 交付前就炸出来。真实语言里重点测德语、俄语这种长的。

>

> 更根本的一条：布局别按字符数写死。控件要能自适应、能换行、能滚动，设计时就得给译文留余量。这是设计标准，测试补不回来。

> **提问：**德语下按钮文字挤成两行，把工具栏撑错位了，这条提给谁？

> **预设回答：**

- 答"提给译者，让他缩短"——有时可行，治标。换俄语还会犯，为排版牺牲译文准确性是本末倒置。

- 答"提给前端，让按钮宽度自适应或者允许换行"——治本的方向。

- 答"提给设计，因为没留余量"——最根本的一层，也是最有价值的判断。

- 收口：好的缺陷单两条都给，建议按钮自适应，短期由译者给个短译法兜住。

> **易错点 / 考点：**德语最长、汉语最精炼是课件原话，可以直接引用。文字扩展的五个受灾点是案例题素材。高频判断"译文太长就让译者改短"——只在极少数场景成立，根本解法是布局自适应。

> **过渡：**8.2 讲完，进 8.3 的技术问题。第一块是数据格式。

【8.29】8.3 软件本地化测试技术（1 分钟）

> 手上拿到一个德语版，先测哪一块？顺序不清，一天就白跑。

>

> 8.3 三块，各管一类问题。8.3.1 数据格式，六项：数字、货币、时间、日期格式、度量衡的单位、复数问题。这一块全都能靠对照来测。同一份数据，在两种区域设置下各显示一次，比对符不符合当地习惯。

>

> 8.3.2 页面显示和布局，三件事。译文长度差异，德语最长、汉语精炼。乱码，双字节语言 GB、BIG5、日文这一类。还有字符索引和排序。这一块靠看：截图、比对、走查。

>

> 8.3.3 配置和兼容性问题，键盘布局、打印机配置、硬件和版本兼容、数据库、热键。这一块靠搭环境：装

> 目标语言的系统、换键盘、插打印机。

>

> 三块合起来，正好是三种发现方式：对照数据、看界面、搭环境。以后接到本地化任务，就按这三个动作走一遍。先拿数据格式对一遍，成本最低，脚本也能帮上忙。再走查界面，这一步最费眼睛。最后在目标环境里跑兼容。

>

> 三块别混着测。混着测的后果是缺陷单里写着"德语版有问题"，开发看了不知道找谁。

>

> 跟前面的关系也说一句。8.1 讲为什么，8.3 讲查什么，8.4 讲怎么组织。三块各有各的用例写法，这一节后面逐个展开。

> **提问：**三块里哪一块最可能被脚本覆盖？

> **预设回答：**

- 答"数据格式，规则明确"——对。喂一组区域设置，比对输出字符串，能写断言。

- 答"乱码可以扫"——部分可以。资源文件里的非法字节序列能扫，界面渲染出来的乱码得靠截图。

- 答"都能自动化"——太乐观。重叠、遮挡、方向错，机器很难判好不好看。

- 收口：格式类尽量交给脚本，界面类靠人眼和截图比对。

> **易错点 / 考点：**三块的名称和内容别混，字符索引排序属于 8.3.2，不属于 8.3.1。三种发现方式是好用的抓手。简答问本地化测试的技术问题有哪些，就答这三块。

> **过渡：**进 8.3.1 之前，先看一个每个项目都会踩的坑：姓名。

【8.30】最常见的问题（2 分钟）

> 注册页上就两个框：姓、名。这两个框坑过的出海产品，比任何复杂功能都多。

>

> 课件给的对照很短：中文写周宇文、朱先生，英文写 Shaomin Zhu、Mr. Zhu。四层差异都在里面。

>

> 第一层，顺序。中文姓在前名在后，英文名在前姓在后。程序如果按第一个词是姓去解析，遇到英文名就全反了。

>

> 第二层，称谓的位置。中文说朱先生，称谓在姓后面；英文说 Mr. Zhu，称谓在姓前面。模板里写某某先生，换到英文就拼成先生挂在后面。

>

> 第三层，称谓怎么选。中文的先生女士，跟英文的 Mr.、Ms.、Mrs. 不是一一对应。Mrs. 还带着婚姻状况的信息，现代用法里很多人更愿意用 Ms.。这一层靠规则表解决不了，得有本地化规范。

>

> 第四层最要命：排序和检索。通讯录按什么排？中文按拼音、按笔画、还是按编码？英文按姓氏的字母序。同一个周，拼音是 Z，排在字母表最后；按编码排位置又变了。用户找一个人，排序错了，功能就等于不可用。

>

> 所以姓名看着是两个文本框，扯着四件事：显示顺序、称谓模板、音译一致、排序规则。这就是本地化的思维，一个普通字段，也要问一遍这规则在别的国家还成不成立。

>

> 它为什么叫最常见的问题？姓名几乎出现在每个系统的第一屏：注册、登录、通讯录、订单、发票、报表。用得最多，出错的机会自然最多。

提问：通讯录里周宇文和 Shaomin Zhu 是同一个人，系统怎么存才不会乱？

预设回答：

- 答"存一个字符串，显示时再拆"——错。一个字符串没法可靠拆出姓和名，排序和称谓全做不了。

- 答"分姓、名、称谓、显示名四个字段"——比较标准的做法，显示名单独存，不用每次拼。

- 答"存拼音就够了"——不行。中文用户在界面上要看汉字，拼音只是排序和检索的辅助。

- 收口：至少三部分，结构化字段、本地显示名、排序键。很多系统是吃过亏才补上的。

> **易错点 / 考点：**四层差异要能说全，只答顺序不同是最常见的丢分点。高频判断"姓名就是一个文本字段"——错，它扯着显示、称谓、音译、排序。拼音、笔画、编码三种排序规则常考细节。

> **过渡：**姓名只是一个字段。8.3.1 要把所有格式类字段系统过一遍，六项。

【8.31】8.3.1 数据格式（3 分钟）

> 同一串 1.234，换个地区读出来能差一千倍。一处读成一千二百三十四，一处读成一点二三四。数字这块出错不报错，只是含义变了，这才是它难查的地方。

>

> 六项，一项一项给判据。数字，两个点：千分位用什么符号，逗号、空格、点、还是不用；小数点用什么符号，点还是逗号。这两个符号在有些地区是互换的。

>

> 货币：符号是什么，放数字前面还是后面，小数留几位，负数怎么写，减号、括号、还是标红。还有舍入规则，按源语言写死，财务报表就对不上。

>

> 时间：十二小时制还是二十四小时制，上午下午怎么标，时分之间用冒号还是点。还有时区，显示本地时间还是服务器时间，跨时区、跨夏令时的时候日期会不会跳。

>

> 日期格式：年月日谁在前，分隔符是什么，月份写数字还是写名称，一周从哪天开始。

>

> 度量衡：长度、重量、面积、温度、速度、纸张尺寸。公里和英里、摄氏和华氏、千克和磅、A4 和 Letter。工业软件、物流、医疗这几类系统里，这是硬需求。

>

> 复数：英语只有单数和复数两态，one apple、two apples。有些语言的复数形式有好几种，不同数量用不同词形。中文、日文基本不分。代码里写一个数量大于一就加 s，放到那些语言里就是错的。

>

> 六项测法统一：准备一组边界数据，在两种以上区域设置下各跑一遍，逐项对。边界数据要带负数、零、很大很小的数、带小数的金额、跨年的日期、夏令时切换那天。

提问：复数不是语法问题吗，为什么单列成一项技术项？

预设回答：

- 答"界面里到处是共 N 条这种拼接，复数错了整句话就错"——最实际的理由。

- 答"有些语言的复数形式不止两种"——准确。不是加不加 s，是选哪种词形。

- 答"复数跟测试没关系"——错。凡是数量加单位的文字都涉及，电商、报表、消息提示里到处都是。

- 收口：复数规则交给本地化框架，别在代码里写数量大于一就加 s。

易错点 / 考点：六项能默写并各举一例。小数点和千分位互换是最经典的案例，要能讲清后果。高频判断"复数就是加 s"——错，多复数形态语言里这条不成立。

过渡：六项都跟区域设置有关。那个设置在哪儿配，影响什么？

【8.32】区域与语言（2 分钟）

控制面板里那两个下拉框，你大概从来没点开过。本地化测试的大半现象，由它们决定。

>

> 一般分三块。第一块，格式。日期怎么写、时间用不用二十四小时、数字的千分位和小数点、货币符号、度量衡单位。这一块只管显示，不动程序逻辑，正好是前面六项的实验台。

>

> 第二块，位置。你在哪个国家或地区。它影响一周从哪天开始、默认纸张、默认度量衡，有些系统还拿它推时区、推天气、推首页内容。

>

> 第三块，管理里的非 Unicode 程序的语言。这一项最关键，也最容易被跳过。它的作用是告诉系统：那些不认 Unicode 的老程序，用哪套编码去解释文字。改这一项通常要重启。很多老软件打开中文全是乱码，

> 根子就在这儿：老程序用系统默认代码页，代码页跟文件编码对不上。

>

> 测的时候做三组实验就够。只改格式不改语言，看格式类的显示变不变。只改语言不改格式，看界面文字变不变。改非 Unicode 程序的语言，看老模块会不会乱码。三组分别对应显示、翻译、编码三层。

>

> 这里有个坑：改完设置要重启或者重新登录才生效，不同版本的 Windows 表现还不一样。没重启就下结论，很容易把正常当成缺陷报上去。

提问：某个老模块打开中文全是乱码，别的模块正常，先查什么？

> **预设回答：**

- 答"查那模块是不是不认 Unicode"——方向对。这类老模块按系统默认代码页解码。

- 答"查系统的非 Unicode 程序语言设置"——准确，这是最常见的原因。

- 答"重装字体"——错。字体解决字形，不解决解码。

- 收口：修法两种，改代码支持 Unicode 治本，调系统默认代码页缓解，但会影响别的程序。

易错点 / 考点：格式、位置、非 Unicode 语言三块各管什么，别混。改设置要重启才生效，这是执行细节，常因为漏掉而误判。非 Unicode 程序语言和代码页的关系是高频考点。

> **过渡：**配置说完了，看这些格式在真实界面上出错的样子。

【8.33】本地化问题 - 数据格式（2 分钟）

财务说金额差了一位，开发说在我这儿显示是对的。这类对话一个月能听到好几次，答案通常在格式上。

>

> 数据格式出错，形态就五类。符号错位：千分位本来用逗号，用了点；小数点本来用点，用了逗号。这一类在数字上最危险，它不报错，只是数值含义变了。

>

> 顺序颠倒：日期写成月日年，目标地区习惯日月年；姓名写成名在前，目标地区习惯姓在前。用户看得出来，

> 看出来就觉得不专业。

>

> 单位没换：公里显示成英里，摄氏显示成华氏，A4 印成了 Letter。打印和报表上后果最直接，直接印错。

>

> 格式串硬编码：整块界面上只有两三个字段不对，别的都对。这通常是那几个字段自己拼了字符串，没走统一的格式化函数。这一类最值得提缺陷，它暴露的是架构问题，不是一处笔误。

>

> 时区处理错误：显示的是服务器时间，跨时区日期差一天，夏令时切换那天的记录时间是乱的。这类特别隐蔽，得在特定日期、特定时区才复现。

>

> 五类的测法还是对照：同一份数据，两种区域设置各截一张图，逐字比。比的时候别凭记忆，手里拿一张目标地区的正确格式参照表，从当地官方习惯或者系统自带格式里取。

>

> 判严重度看一条：它有没有流进数据层。只在界面上显示错的，和已经写进库、发给下游的，不是一个级别。

提问：界面上只有两三个字段日期格式不对，别的都对，根因是什么？

预设回答：

- 答"这两个字段特殊处理了"——对。通常是有人图省事，自己拼了字符串。

- 答"翻译错了"——不太可能。日期格式是格式化函数算出来的，不是翻出来的。

- 答"字段类型不对，存的是字符串"——很专业，这是根因层面：字符串就没法格式化。

- 收口：这条缺陷的价值不在改这两个字段，在全项目搜一遍还有多少地方在拼格式串。

易错点 / 考点：五类形态要能认出来。局部格式不对往往指向硬编码，这是案例题的高分点。严重度看它有没有流进数据层，有没有影响金额或时间。

过渡：数据格式说完，进 8.3.2，从文本长度开始。

【8.34】8.3.2 页面显示和布局（3 分钟）

中文界面截图和德语界面截图并排放在一起，差别一眼就看出来。课件用三句话把这一块说完了。

>

第一句，德语最长，汉语比较精炼。布局问题的总根源在这句里。德语靠复合词承载信息，一个词能很长；中文一字一义，同样的意思用的字符少。所以中文界面译成德语，几乎必然装不下；反过来英文译成中文，往往留一大片白。测试的含义很直接：拿最长的语言和最短的语言各测一遍，中间的自然会过。

>

第二句，乱码，括号里是 GB、BIG5、日文这一类。这三个代表三类双字节编码：GB 是简体中文国标，BIG5 是繁体中文，日文是另一套。共同点是一个字符两个字节，也共同面临编码不一致就乱码。这句还提示了真实情况：一个系统里常常同时存在好几种编码。一份 UTF-8 数据，一份遗留的 GB 数据，还有一份日文系统导出的文件。

>

第三句，字符索引、排序。索引是按什么建目录：拼音、笔画、部首、字母。排序是按什么比大小：不同语言的字母表顺序不同，还有变音符号、大小写、数字混排。中文环境里就已经很麻烦，拼音序和笔画序结果不一样。多语言下更难，一个列表里中文英文日文都有，按什么排？

>

三句归一处：布局测试就问三件事，长度够不够、编码通不通、顺序对不对。对应的手段也是三类：截图比对、编码扫描、排序结果核对。

提问：一个列表里同时有中文、英文、日文的名字，按什么排？

预设回答：

- 答"统一按编码排"——技术上稳定，但用户看不懂：汉字按编码排，既不是拼音也不是笔画。

- 答"按当前界面语言的规则排"——主流做法，中文界面按拼音、英文界面按字母，符合预期。

- 答"随便排，用户会自己搜"——错。排序是检索的基础，排序乱就是找不到东西。

- 收口：排序规则要写进需求，还要写清混合语言怎么办，不然开发和测试各按各的理解做，必吵。

易错点 / 考点：德语最长、汉语精炼是课件原话。GB、BIG5、日文分别对应简体、繁体、日文，别搞混。索

引和排序是两个概念，考试常混。

> **过渡：**装不下的时候，界面上到底长什么样？

【8.35】示例：不能完全显示（2 分钟）

缺陷单上写界面显示不全，开发回你一句截个图。截图发过去，他说这不算 bug，字能看见。争到这儿，说明缺陷没写清形态。

>

> 不能完全显示有四种形态。硬截断：字直接被切掉，末尾没有提示。用户看到半句话，还不知道后面有内容。四种里最差。

>

> 加省略号：截断了但留了省略号，用户知道还有内容，看不到。比硬截断好一点，前提是有悬停提示，或者点开能看全文。没有的话，用户只能猜。

>

> 撑破容器：字没被切，但把控件撑变形了。按钮拉成长条，对话框被顶宽，工具栏换行把下面所有内容都推下去。这种连带一片，一个控件变宽，整行都乱。

>

> 压盖：文字盖住旁边的图标或者别的文字，两层叠在一起，两边都读不了。

>

> 严重度也按这个看。压盖和撑破最重，它们影响别的元素；硬截断次之；加省略号最轻，前提是全文能拿到。

>

> 测的时候有个细节别漏：同一个界面，换分辨率、换字号、换 DPI，表现可能完全不同。你在自己那块宽屏上看着正常，用户在笔记本上就是截断。所以布局测试要在目标地区常见的分辨率下跑，不能只在开发机上瞄一眼。

>

> 截图还有个规矩：源语言版和目标语言版截同一个界面，两张并排放进缺陷单。开发一眼就知道差多少，不用来回问。

提问：文字加了省略号，算不算合格？

> **预设回答：**

- 答"算，至少有提示"——不完全算。要看全文能不能拿到：悬停、点开详情、或者能拖列宽。

- 答"不算，让控件自适应才好"——治本的要求。现实里列宽固定是常态，省略号加完整提示是常见折中。

- 答"看字段，关键信息不能省"——很专业。金额、日期、编号被截断，后果比描述性文字重得多。

- 收口：判合格的标准是信息能不能被用户完整拿到，不是看起来有没有提示。

> **易错点 / 考点：**四形态和严重度排序。同一界面在不同分辨率、DPI 下表现不同，这是执行要点，常被漏。缺陷单要双语截图并排，这是实操规范。

> **过渡：**装不下是一种问题，读不出来是另一种。

【8.36】示例：乱码（2 分钟）

> 屏幕上冒出锟斤拷，有点经验的人已经知道大概哪儿错了。乱码不是一种病，是三种。

>

> 第一种，成串的怪字符。原来的中文位置变成一堆看不懂的符号。根因几乎总是编码不一致：数据是 UTF-8，程序按 GBK 解了，或者反过来。特征是同一份数据换个入口、换台机器就正常。

>

> 第二种，方框或者豆腐块。每个字的位置一个空心方框。这不是解码错，是字体缺字形：编码解对了，可这个字体里没有这个字符的形状。特征是换个字体就好了。

>

> 第三种，半个字或者问号。一个汉字只显示一半，或者干脆变成问号。这通常是截断造成字节断裂，或者经过一次表示不了就替换成问号的有损转换。特征是这个字段往往有长度限制。

>

> 三种现象，三种根因，三种修法。第一种改编码声明或者转换逻辑；第二种装字体、换字体；第三种改截断方式，按字符截，不按字节截。

>

> 所以提缺陷要写到层次上。报一句界面有乱码，开发只能猜。写成这个字段在 UTF-8 数据源下按 GBK 解码，建议统一成 UTF-8。他十分钟就改完。

>

> 还有一条：乱码往往不是全乱，是局部乱。只有走过某条特定路径的数据才会错。复现的时候把链路记清楚：数据从哪儿来、经过哪些环节、在哪一步被转了码。

提问：同一段文字，网页上正常，导出成文件再用记事本打开就乱码，怎么解释？

> **预设回答：**

- 答"文件编码和网页编码不一样"——对。网页有编码声明，文件没有，记事本只能猜。

- 答"记事本太老"——事实层面也对。老版本对编码的自动识别很弱，容易猜错。

- 答"文件坏了"——错。文件完好，是解释方式错了。

- 收口：导出功能要保证文件里带编码声明，或者统一用兼容性最好的编码。这条能直接写成用例。

易错点 / 考点：三种现象和三种根因的对应关系要能背。局部乱码的复现要记数据链路，这是实操要点。高频判断"乱码装个字体就好"——只对第二种现象成立。

> **过渡：**还有一类问题不在文字上，在图片和控件里。

【8.37】示例：其它（2 分钟）

界面走了三遍，能找的问题都找了，上线照样被投诉。漏掉的通常不在文字上。

>

> 不属于长度、不属于乱码的问题，收起来有四类。第一类，图片里的文字。图标上写着 Save，按钮背景图里印着 Submit，帮助文档的截图里全是英文界面。这类文字抽不进资源文件，只能重画图。测试时要专门去翻图片：图标、徽标、示意图、文档截图。这是最容易被漏的一类，因为眼睛全盯着文本。

>

> 第二类，图标的隐喻。邮筒、手势、动物、颜色、节日元素。同一个图标在不同的文化里含义能差很远。这一类不是显示错误，是文化不适。

>

> 第三类，隐含的格式假设。地址栏按省市区的顺序排，姓名字段默认两个字就是姓名，邮编只留六位数字。这些假设在源语言环境里天经地义，换个国家就全崩。测的时候把每个输入框问一遍：这个格式在别的国家还成立吗。

>

> 第四类，跟语言绑在一起的行为。大小写转换，有些语言没有大小写。首字母大写，有些语言不这么用。连字符断词，只对拉丁语言有效。还有文字方向。还有按字母分组的联系人列表，中文按拼音分组，日文按假名。这些都算行为层面的本地化。

>

> 一句话：本地化不只是文本。走查的时候眼睛分层看，文字一层、图片一层、控件一层、行为一层。

> **提问：**按首字母分组的联系人列表，在中文环境下怎么改？

> **预设回答：**

- 答"改成按拼音首字母分组"——主流做法，中文用户也熟。

- 答"按笔画或者部首"——可行，认知成本高，实际产品很少这么做。

- 答"不用改，中文也能按编码分组"——错。编组分出来的组名，对用户毫无意义。

- 收口：这类跟语言绑定的行为，改动量往往比翻译大得多，改的是逻辑，不只是文本。

> **易错点 / 考点：**图片里的文字抽不出来、必须重画，这是高频考点。隐含格式假设的例子要能举：地址、姓名、邮编、电话。常考判断"本地化就是把文字翻一遍"——这四条就是反驳。

> **过渡：**把这些零散的点收成检查项，就是下面六条。

【8.38】UI 验证的细节（4 分钟）

走查报告里最没用的一句是界面基本正常。什么叫基本正常？六条细节，每条都能直接变成检查项。

>

第一条，控件互相重叠，或者排列间隔不均衡。重叠是硬伤，两个控件压在一起，用户点不准。间隔不均是看着别扭，一处挤一处空。根因多半是布局用了绝对坐标，译文一变长就挤到别人地盘上。

>

第二条，文字遮挡图像、文字超过边界、控件里字符没显示全。三种都是装不下，但得分开：遮挡是层级问题，超边界是容器问题，字符不全是裁剪问题。修法不一样。

>

第三条，文字方向。课件点了两种从右往左的语言：希伯莱文和阿拉伯文。要验的是整段方向对不对、控件顺序有没有镜像、标点位置对不对、数字在右往左的段落里摆得对不对。

>

第四条，左右对齐。课件给了两个例子：阿拉伯文右对齐；中文和英文有区别，中文段落开头空两个字，英文开头不空。这个细节特别说明问题。首行缩进不是审美偏好，是排版规则，按语言走。程序统一加了缩进，英文段落看着就怪；中文段落不缩进，读着也不顺。

>

第五条，连字符。对多数拉丁语言有效，对东方语言一般无效。连字符断词是为了行末把长单词拆开，拉丁语言按音节拆合理。中文日文没有词间空格，可以在任意字之间换行。排版引擎给中文加连字符，就是错的。

>

第六条，拉丁语言的大小写，多字节语言的乱码。全部大写这种样式处理，碰到中文、日文、阿拉伯文会怎样？乱码还是前面那三种现象。

>

六条盖住的是界面上看得见的全部：位置、边界、方向、对齐、断行、字符。

提问：中文首行空两字、英文不空，这个差别由谁保证，翻译、设计，还是代码？

> **预设回答：**

- 答"翻译时在段首加两个空格"——最常见的土办法，很脆。换字号、换字体、复制粘贴都会错位。

- 答"由样式控制，按语言设首行缩进"——正确做法，样式跟着语言走。

- 答"无所谓，看不出来"——错。排版规则是本地化的基本礼貌，专业用户一眼看出这不是本地做的。

- 收口：样式也要国际化。同一个样式类，在不同语言下可以有不同的缩进、行高、字体。

> **易错点 / 考点：**六条按位置、边界、方向、对齐、断行、字符归类。中文首行缩进两字、英文不缩进，是课件明确给的细节，适合出选择题。高频判断"连字符断词对所有语言都适用"——错，对东方语言一般无效。

> **过渡：**界面看完了，还有一层在环境里。

【8.39】8.3.3 配置和兼容性问题（3 分钟）

> 同一份安装包，自己机器上好好的，客户那台一按快捷键就串了。这类问题不在代码里，在环境里。

>

> 课件三句话。第一句，配置性包括键盘布局设计，它是语言依赖性最大的硬件，还有打印机配置等。注意这个说法，键盘。为什么是键盘？因为它是物理的。德语键盘的 Z 和 Y 位置对调，法语键盘是 AZERTY，阿拉伯语键盘上还有专门的键。软件要是按物理位置或者按字符绑快捷键，换块键盘就可能失效，或者按 A 出来 B。打印机也算配置：纸张尺寸、默认边距、驱动语言，都会影响输出。

>

> 第二句，兼容性包括与硬件的兼容、与上一版本的数据兼容、跟别的本地化软件的兼容。三类要分清。硬件兼容，在目标地区常见机型上能不能跑。版本兼容，从上一版升上来，数据和设置会不会坏。跟别的本地化软件兼容，同机装了别的语言包、别的输入法、别的办公套件，会不会打架。第三类最容易被忽略，现实里最常见，用户机器上从来不止你一个软件。

>

> 第三句，关注一些点，比如数据库问题、热键。数据库的字符集、排序规则、字段长度，跟本地化数据对不对得上。一个 GBK 排序规则的库塞进 UTF-8 的数据，轻则乱码，重则截断。热键要重新分配，还要躲开系统保留键和别的软件占用的组合。

>

> 三句合起来：配置看键盘和打印机，兼容看硬件、版本和邻居软件，细节看数据库和热键。这一块没有巧办法，只能搭环境、真机跑。

> **提问：**键盘为什么被叫成语言依赖性最大的硬件？

> **预设回答：**

- 答"布局不一样，键位和字符的对应关系不同"——核心原因。

- 答"键盘上印的是当地字母"——也算一层。阿拉伯语、俄语键盘上印的就是当地字母。

- 答"键盘就是输入设备，跟语言无关"——错。同一台电脑换块布局不同的键盘，敲同一个物理键出来的字符可能完全不同。

- 收口：测快捷键不能只在自己键盘上按一遍，得在目标语言的键盘布局下再按一遍。

> **易错点 / 考点：**键盘是语言依赖性最大的硬件，这是课件原话，常出填空或判断。兼容性三类要能说全：硬件、上一版本数据、别的本地化软件。数据库的字符集和排序规则是高频细节。

> **过渡：**具体怎么测？下面三个场景都很工程。

【8.40】配置和兼容性测试（2 分钟）

> 有人问，机器上装了德语语言包，界面全是德语，还要不要再开一台德语系统？要。

>

> 课件给了三个切入点。第一，本地化操作系统。测本地化版本，最好在目标语言的操作系统上跑，不是在自己机器上装个语言包。语言包只改界面语言。系统的默认代码页、区域格式、字体集合、输入法，甚至某些接口的行为，可能一个都没变。在中文系统上装个德语包测出来的通过，不可靠。这一条在所有本地化测试规范里都排第一条。

>

> 第二，Non-Unicode 数据路径。程序处理非 Unicode 编码数据时走的那几条路。读一个 GBK 文本，导一个 BIG5 的 CSV，访问一个老数据库。这类路径在现代代码里多半是历史遗留，最容易出乱码和截断。测法很直接：准备一批不同编码的数据文件，逐个导入，看结果。

>

> 第三，欧洲系统本地化 OEM 和 Windows 的 ANSI。同一个 Windows，不同地区的 OEM 版本，默认的 ANSI 代码页可能不一样。西欧 1252，中欧 1250，中文 936，日文 932。所谓 ANSI 不是一种编码，是一堆代码页的统称。程序如果依赖系统 ANSI 代码页，它在不同地区的机器上行为就不同。同一份程序在不同国家表现不一样，技术根源就在这儿。

>

> 课件还给了 W3C 的一篇文章，Migrating to Unicode。W3C 的技术文章，讲怎么把一个老的单字节系统安全迁到 Unicode。数据怎么迁，代码怎么改，迁移中怎么不丢数据。看完这一篇，8.3.3 的问题基本都能对上号。

>

> 三点收一句：真环境的核心差异，一半在操作系统，一半在编码。

提问：中文 Windows 上装德语语言包，能不能替代德语版 Windows?

> **预设回答：**

- 答"能，界面都德语了"——错。界面语言只是表层，代码页、字体、输入法、区域默认值都可能没变。

- 答"不能，但可以先当筛子用"——务实。语言包环境能发现一部分翻译和布局问题，编码类必须上真系统。

- 答"看测什么，测翻译够用，测编码不够"——完全正确，这就是分层测试。

- 收口：计划里要写清哪类用例在语言包环境跑，哪类必须上目标语言系统。

易错点 / 考点：ANSI 不是一种编码，是一堆代码页的统称，这点极易考。三个切入点的名称要能说出。高频判断"装语言包等于本地化环境"——错。

> **过渡：**8.3 到此收尾，进 8.4，把本地化版本当成一个完整软件来测。

【8.41】8.4 本地化功能测试（1 分钟）

德语版装上去，菜单翻对了，按钮翻对了，日期写成 31.12.2026。看着能交了。有人顺手点一下导出，导成

> Excel 再导回来，日期全乱。这个错，哪张清单都没列。

>

> 8.3 走的是另一条路：查格式、查布局、查兼容，一类一类过。像拿着清单勾，勾得干净。可清单只管单项，管不了前后接不接得上。

>

> 8.4 换了法子。把本地化版本当成一个完整软件。让一个真用户从头走：安装、登录、用功能、打印、导出、看帮助、升级。一路走到尾，卡在哪一步，缺陷就在哪一步。

>

> 为什么非走一遍？用户是连贯的，清单是分开的。第 5 项没出事，第 7 项也没出事，可第 5 项吐出来的东西喂进第 7 项，就出事。这类缺陷只活在链路里，单项检查永远碰不到。

>

> 两节的分工也就在这儿。清单管覆盖，保证每一项都看过；场景管连通，保证环和环咬得住。谁也替不了谁。清单漏的是组合，场景漏的是细节。成熟的队伍两样都做，先扫清单，再走场景。

>

> 还有一个现实原因。清单勾完了，纸上留着勾，谁都能看见你查过。链路走一遍，走没走全凭自述，所以这一步最容易省。可省下的正是别人栽跟头的地方。反过来说，只走链路不扫清单，漏的是单项细节，一页翻错一个词也看不出来。

>

> 这一节的关键词是集成。译文全对，只说明每一项对；整条业务链在目标语言环境里跑得通，才叫能交付。差一环，用户就停在那儿，前面的功夫全白费。

提问：清单式测试和场景式测试，哪个更容易漏缺陷？漏的是哪一类？

预设回答：

- 答"清单容易漏，因为清单是有限的"——对，清单只能盖住想得到的。

- 答"场景容易漏，因为没有明确判据"——也有道理，场景测试靠经验和想象力。

- 答"两种都会漏，漏的东西不一样：清单漏组合，场景漏细节"——这个最完整。

- 收口：所以成熟的做法是叠着用，先按清单扫覆盖，再按场景走连通。

> **易错点 / 考点：**① 8.3 与 8.4 的定位差别（清单 vs 场景）是读懂本章结构的关节；② 高频判断"本地化测试就是检查界面翻译"，这两节都在反驳它；③ 案例题常给"导出再导入出错"这类跨环节缺陷，考点就是组合缺陷。

> **【思政落点 5】**：导出再导入这条链上出的错，每一项单独看都合格，没人挑得出毛病。缺陷就躲在这种"人人过关、合起来出错"的地方。这一节要求学生从用户走完全程，考的是不肯只交自己那一份合格答卷的习惯。工序之间那道缝，总要有人愿意替下一道工序多看一眼。

> **过渡：**进入具体工作，这一节分三块，还对应两种行业里通行的组织形式。

【8.42】本地化功能测试的具体工作（3 分钟）

> 一份本地化版本要过几拨人的手？语言一拨，功能一拨，界面一拨。这三摊子活，很少落在同一个人身上。

>

> 先看语言。查的是译文本身：漏译、错译、术语、语气、符号。再看功能。查业务在目标语言环境下正不正常：输入、处理、输出、打印、导出。还有一摊，美观和界面：排版、对齐、截断、方向、图标。

>

> 这么分为什么好用？它正好对上三类人。语言交给译审或本地语言专家，功能交给测试工程师，界面交给 UI 或前端。人分对了，返工就少。让测试工程师去改译文语气，他改不动；让译审去判断按钮溢出不溢出，那也不归他管。

>

> 组织形式有两种。第一种，In-country testing，本土测试，也叫在当地测。把测试放到目标市场做：当地机器、当地网络、当地系统版本、当地用户习惯，最好当地人自己上手。

>

> 它值钱在一个真字。当地网速、当地支付习惯、当地常见机型、当地人的操作直觉，你在国内模拟不出来。代价也明摆着，贵，协调还难。

>

> 第二种，Translation verification testing，TVT，翻译验证测试。窄而深，只验翻译。语言专家拿双语对照的界面逐条看：准不准，合不合语境，术语统一不统一。工具一般给三样：双语对照、术语库比对、上下文截图。

>

> 那 TVT 跟普通的翻译审校差在哪？差在上下文。同一句话，摆在按钮上和摆在提示框里，译法可能就不一样。脱离界面的审校看不到语境，看不到长度限制，也发现不了占位符写错。它顺手还能撞出截断，那是布局的毛病。

>

> 两者什么关系？TVT 保字对，本土测试保用得对。钱紧就先 TVT，便宜，能并行，把语言问题清干净；再上本土测试做最后把关，贵，可它替不掉。

提问： TVT 和普通的翻译审校，差在哪一步？

预设回答：

- 答"TVT 是在软件界面上看，审校是看文档"——很准。关键是它在上下文里看。

- 答"TVT 还要查术语一致性，也看有没有截断"——补得好，TVT 常连带发现布局问题。

- 答"没区别，都是看翻译"——错。脱离界面的审校看不到语境、看不到长度限制、也发现不了占位符错误。

- 收口：TVT 是在真实界面里验翻译，这一点它不可替代。

易错点 / 考点： ① 三块内容与三类执行者的对应关系；② In-country testing 与 TVT 的英文全称、中文译名、定位差别都要说得出；③ 高频判断"TVT 可以替代本土测试"——错，两者互补，一个保字对，一个保用得对。

过渡： 这一节收尾落到 L10N 功能测试的三项内容：集成、索引排序、联机文档。

【8.43】L10N 功能测试（3 分钟）

界面翻完了，译文也校过了，能不能收工？收不了。

>

> 这一节有三项内容。第一项，集成测试。本地化版本不是孤岛。它得跟操作系统、数据库、打印机待在一起，

> 还得跟外部系统接口、别的语言版本待在一起。典型事故有三类。装两个语言版本互相覆盖；语言包和主程序版本对不上；跟本地办公软件换文件时编码乱。

>

> 第二项，索引和排序。8.3.2 讲的是原理，这里看它能不能用。搜索搜不搜得到，按字母、拼音、笔画建的索引对不对。排序合不合当地习惯，分页和跳到某个字母点得动点不动。通讯录、文档管理、电商商品分类，全靠这个。排序一错，搜索等于废了。

>

> 第三项，联机文档的功能测试。是功能，不是内容。帮助能不能打开，目录能不能跳，搜索能不能用，索引链接有没有死，PDF 里的书签和字体正不正常。内容准不准归翻译验证，文档能不能用归功能测试，提缺陷要分开提。

>

> 还有一句得念一遍原文。它不仅要求查看用户界面，而且要对文件保存、打印这类功能做测试。特别要盯语言环境特定的组件，比如时间、日期格式，还有文字处理。这句在纠什么？纠的是"本地化就是看界面"这个偏见。

>

> 界面之外出事，你往往看不见。存下来的文件名对不对，导出文件的编码对不对。打印的纸张和边距对不对，复制粘贴到别的软件里会不会乱码。这些地方没有界面，可用户干活就落在这儿。

>

> 所以给一句能直接用的。保存、打开、打印、导出，加上复制粘贴和发邮件。这六个动作，每个都拿目标语言的数据做一遍。

>

> 打印为什么单拎出来说？纸张尺寸不一样，A4 换 Letter 就错位。打印走的是另一套字体和渲染，中文可能出方框。它牵着纸张、边距、字体、编码、驱动，每一种都可能跟地区有关。界面看着好、输出却错，打印最典型。

提问：为什么"打印"在本地化测试里要被单独强调？

> **预设回答：**

- 答"纸张尺寸不一样，A4 和 Letter 会错位"——对，这是最直接的原因。

- 答"打印出来的字体可能不对，中文变方框"——也对，打印链路是另一套字体和渲染。

- 答"打印跟本地化没关系"——错。纸张、边距、字体、编码、驱动，都可能跟地区有关。

- 收口：界面看着好、输出却错，打印就是这个场面，8.3.3 把打印机配置列出来，道理在这儿。

> **易错点 / 考点：**① 三项内容要说得出口；② 那段引文是全章高频引用句，"界面之外的功能"是核心观点；③ 高频判断"本地化测试只需要看界面"——这一节就是现成的反例。

> **【思政落点 6】**：保存、打开、打印、导出这几步，界面上没有入口图，检查单上也常整段空缺，最容易一省了之。这一节偏把它们和语言环境特定组件绑在一起：时间、日期格式、文字处理，一错就是用户手里的文件打不开、报告印不出来。测试这份活的分量就在没人盯的地方：不走这一遍，交出去的是自己的侥幸。

> **过渡：**本章的内容讲到这儿，我们收一下，把整章压成三句话。

【8.44】本章小结（3 分钟）

> 整章走完，我把三句话摆在这儿，抄下来就够用。

>

> 第一句：软件国际化是研发上的多工程问题，设计和编程要守 I18N 的规范和要求。多工程不是很多工程，是说它横跨设计、编码、构建、文档，贯穿整条线。它不是能事后补的功能。这一句对着 8.1：i18n 属于 Engineering，做一次，所有语言都受益。

>

> 第二句：国际化测试要验八件事。双向识别、语言切换方式、多字节和单字节混排、输入法编辑器。大小写转换、快捷组合键、词序问题，还有无硬编码。这八项是 8.1 那十项测试点的浓缩版，落脚点在最后三个字。

>

> 为什么落在无硬编码？因为绝大多数国际化缺陷从这个根上长出来。日期格式写死、货币符号写死、提示语拼在代码里，语言一换全崩，还得回头改一遍代码。

>

> 第三句：本地化测试涉及数据格式、UI 显示和布局、配置和兼容性问题、翻译内容。四类，对着 8.3 和 8.4。再加上 8.1.5 的六类测试工作、8.2 的翻译验证，执行面就齐了。

>

> 三句合起来是一张责任图。研发负责能做到，测试负责验证做到了，本地化团队负责做得好。出事先定位是谁那一段，别一上来就吵。

>

> 给你一个三问自检，考试和面试都用得上。这个功能有没有把地区相关的东西硬编码？同一份数据在目标地区显示成什么样？在目标环境里，整条业务链跑得通吗？三问都过了，本地化质量基本靠得住。

>

> 只许留一条用例，留哪条？看业务风险。金融产品留格式和金额，出海消费品留布局和翻译。资源永远不够，会取舍才是专业。

>

> 还有一处常混，答题前先对号：8.1 的八条能力、五条标准、八步流程、六类测试。综合题按这条线答——i18n 先行，伪翻译，TVT，本土测试，发布验证。

提问：如果只允许你保留一条本地化测试用例，你留哪一条？为什么？

预设回答：

- 答"切换语言后逐页走查"——覆盖面最广，翻译、布局、方向、编码一条用例全压上。

- 答"用德语在德语系统上跑一遍主流程"——同样值钱，长度、格式、环境一次压到。

- 答"随便留一条，反正测不完"——这个心态要纠。资源永远不够，取舍才是专业能力。

- 收口：看业务风险。金融产品留格式和金额，出海消费品留布局和翻译。

易错点 / 考点：① 三句话要背得住，特别是"多工程"和"无硬编码"两个词；② 本章四组数字容易混：8.1 的八条能力、五条标准、八步流程、六类测试，答题前先对号；③ 高频综合题"如何组织一个多语言版本的测试"，按 i18n 先行 → 伪翻译 → TVT → 本土测试 → 发布验证这条线答。

过渡：课上的内容到这儿，最后留几分钟，把可以继续往下挖的地方指给你。

【8.45】参考资源（1 分钟）

> 课上讲完了，课下想接着往下挖，去哪儿找？

>

> 五个入口，我按用处排。

>

> W3C 的 International 站点，管字符编码、语言标签、文字方向、本地化最佳实践。8.1.2 和 8.3.3 里那些标准，很多在这儿有写给开发者看的解释。只看一个地方，就看它。

>

> Sakai 项目的 I18N 主页，是开源教学系统的国际化资料。它值在工程实践：资源怎么抽出来，多语言怎么管，翻译流程怎么走。看真项目怎么干，比看规范接地气。

>

> GALA Global 是全球化与本地化行业协会。产业分工、质量标准、术语、职业路径，都在那儿。想知道本地化是门生意，看它。它是协会，不是标准组织，这两个别混。

>

> MSDN 上 Kieran 的博客，长期写国际化和编码问题。特点是具体，一个乱码案例能写清来龙去脉。这类东西适合手里正卡着一个问题的时候翻。

>

> Java 官方的国际化入门文章，讲资源包、区域设置、格式化类。要做实验从这儿入手最快。Java 这套框架最经典，看懂了它，别的语言是同一套思路。

>

> 用法一句话：W3C 打底，Sakai 看流程，GALA 看行业，博客查案例，Java 做实验。别想一次看完，遇到问题再查，效率最高。

>

> 翻的时候多看一层：它讲的是原理、流程，还是案例。别拿一篇博客当标准，也别拿一份标准当操作手册。标准写的是应当怎样，手册写的是按哪几个按钮，两样混着用，写出来的东西两头都不到位。

>

> 要给团队写入门材料，先看哪个？我的建议是 W3C 加 Sakai，再加一份自己项目里的真案例。一份原理，一份实践，一份自己的现场。

提问：如果你要给团队写一份"本地化测试入门材料"，你会先看这里的哪一个？

预设回答：

- 答"W3C，最权威"——权威，可偏原理，写入门材料容易太硬。

- 答"Sakai，完整项目的过程都在里面"——很实用，能直接看到测试计划怎么组织。

- 答"都不用看，我自己总结"——不建议。这几条路别人走过，坑也写下来了。

- 收口：一份原理，加一份实践，再加自己项目里的案例，这个组合最顶用。

易错点 / 考点：① 五个资源各自的性质要对得上：标准组织、开源项目、行业协会、技术博客、语言框架文档；② 考试一般不考网址，但会考"本章推荐了哪几类参考资料"；③ GALA 是行业协会，别当成标准组织。

过渡：这一章到这儿收住，最后说几句收尾的话。

【8.46】感 谢 聆 听（1 分钟）

很多年以后，这一章大概只剩一句话留在你脑子里。

>

> 第一句，难点不在技术，在换位。你要能站到一个德国用户、一个阿拉伯语用户、一个美国用户的位置上。问一句：这样写，我看不看得懂，用不用得顺。这个本事比会写脚本难，也更值钱。换不过来，做出来的东西就只服务自己人。

>

> 第二句，越早做越便宜。国际化是设计阶段的事，本地化是发布前的事。设计评审时多问一句"这个格式硬编码了吗"，可能省掉后面几周返工。软件工程里能说几乎没有例外的规律不多，这一条算一个。

>

> 第三句，细节就是专业。一个编码参数，一个首行缩进，一个称谓位置，一个复数词形，单看都不起眼。可

> 用户看到的全部，就是这些不起眼的东西。它们凑在一起，就是别人对这个软件的印象，改一个星期的架构也换不回来。

>

> 三条主线收一下：i18n 的能力，l10n 的执行，文化合规。三样在脑子里要连成一张图，各记各的，考试一综合就散了。

>

> 从下一章起换个视角，从测一个软件，转到管一个测试项目。上面几章教你怎么把东西测对。下面就问你怎么把测这件事管顺：人要几个，进度怎么排，风险什么时候冒头。测对一个版本，和让一堆人在期限里都测对，是两种本事。

>

> 带走一个习惯就够：看到一个理所当然的设计，先问一句"换个地方还成立吗"。

>

> 页面最底下那行是学校和名字，署名，不用抄。谢谢大家。

> **提问：**这一章里，哪一个知识点最可能在你将来的工作中直接用到？

> **预设回答：**

- 答"字符集和乱码的排查思路"——很实用，几乎每个项目都会撞上。

- 答"文字扩展和布局自适应"——同样实用，做过出海项目的人都认。

- 答"这些用不上，我们只做国内市场"——不一定。繁体版、少数民族语言版、多终端多分辨率，方法是一样的。

- 收口：这一章给的是在差异里保住一致的本事，换到哪儿都用得上。

> **易错点 / 考点：**① 本章三条主线（i18n 能力、l10n 执行、文化合规）要能在脑子里连成一张图；② 复习先背四组数字：八条能力、五条标准、八步流程、六类测试；③ 案例题通用答法：先判层次（i18n 还是 l10n），再判类型（格式、布局、编码、文化），最后给建议。

> **过渡：**本章结束，下一章进入测试项目管理，我们从"怎么测"走向"怎么管"。

四、思政落点清单（共 6 处）

序号	所在页	落点主题
1	8.11	字符集这一块最有意思的地方是，它出错的时候程序一切正常。没有异常，没有崩溃，日志
2	8.20	把缺陷写清楚，是件没人监督的事。"界面显示异常"六个字也能交差，写清环境、现象、
3	8.21	八步里最不显眼的是第一步配置管理。它不产出任何看得见的东西，却决定后面七步有没有
4	8.27	文化这一栏考的是尊重和合规。有些内容我们看着平常，在别人那里是禁区；有些表述国内
5	8.41	导出再导入这条链上出的错，每一项单独看都合格，没人挑得出毛病。缺陷就躲在这种"人
6	8.43	保存、打开、打印、导出这几步，界面上没有入口图，检查单上也常整段空缺，最容易一省